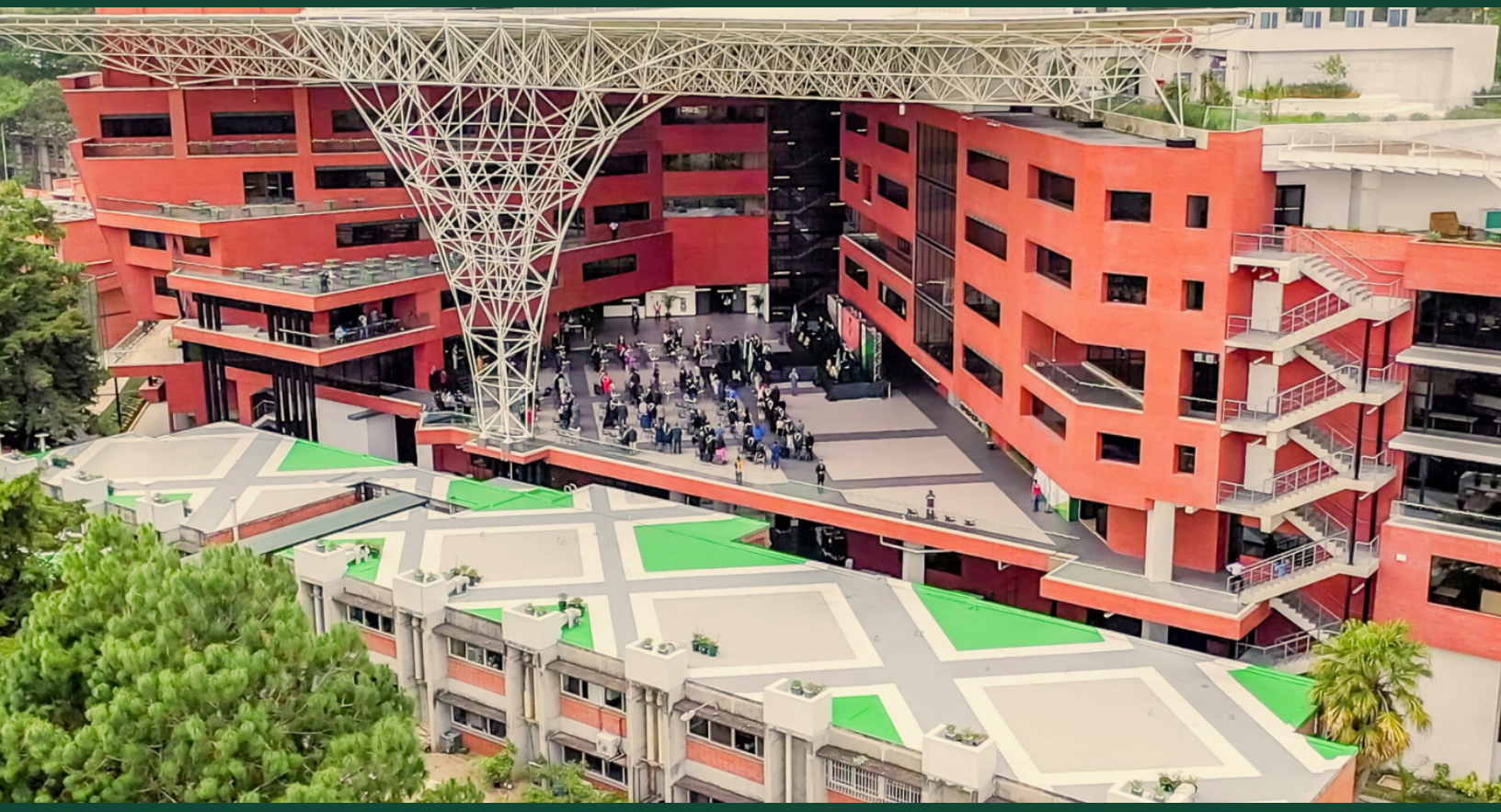
Aplicación de algoritmos de aprendizaje profundo a señales bioeléctricas para la identificación de segmentos de interés en el estudio de la epilepsia

Santiago Sánchez Castañeda



UVG | UNIVERSIDAD DEL VALLE DE GUATEMALA

UNIVERSIDAD DEL VALLE DE GUATEMALA
Facultad de Ingeniería

Aplicación de algoritmos de aprendizaje profundo a señales bioeléctricas para la identificación de segmentos de interés en el estudio de la epilepsia

Trabajo de graduación presentado por Santiago Sánchez Castañeda para optar al grado académico de Licenciado en Ingeniería Mecatrónica

Guatemala,

2025

UNIVERSIDAD DEL VALLE DE GUATEMALA
Facultad de Ingeniería

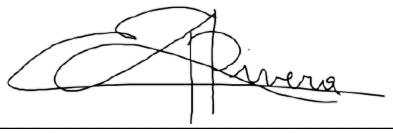
Aplicación de algoritmos de aprendizaje profundo a señales bioeléctricas para la identificación de segmentos de interés en el estudio de la epilepsia

Trabajo de graduación presentado por Santiago Sánchez Castañeda para optar al grado académico de Licenciado en Ingeniería Mecatrónica

Guatemala,

2025

Vo.Bo.:

(f) 

Dr. Luis Alberto Rivera Estrada

(f) 

M.Sc. Carlos Esquit Hernández

Fecha de aprobación: Guatemala, 20 de noviembre de 2025.

Prefacio

Aprovecho este espacio para dar mi agradecimiento a mi familia y amigos por todo su apoyo y acompañamiento durante mi formación, su influencia en mi desarrollo como adulto y el amor y paciencia que me han dado.

Gracias a mi mamá, Ana Patricia, por ser mi mayor ejemplo de una persona responsable, dedicada y perseverante. Gracias por tu tiempo, por escucharme y aconsejarme, por acompañarme con tanto cariño y tu apoyo constante durante todas las etapas de este camino. Tu fortaleza y perseverancia ante cualquier situación han sido inspiración continua para mi.

Gracias a mi papá, Lisandro, por siempre impulsarme a ir más lejos, no conformarme y aprovechar y fomentar mi creatividad. Te agradezco las experiencias de las que me has hecho parte que han ampliado mis horizontes, la confianza para que tome riesgos y tu ejemplo como persona creativa y de pensamiento crítico. Mucho de lo que soy hoy te lo debo a tu forma de pensar y a tu cariño.

A mi hermana Sara, a quien admiro profundamente. Su sensibilidad, esfuerzo, disciplina y ambición me han inspirado y enseñado el valor de seguir mis convicciones y pelear por lograr lo que quiero. Ojalá que todo lo que te caracteriza te lleve tan lejos como vos te lo permitas y que en el camino encuentres tanto felicidad como éxito.

A mis abuelos, Raúl, Patricia, Yoli y Luis, a quienes quiero enormemente y que siempre han estado pendientes de mi crecimiento. Su ejemplo de perseverancia y su interés genuino po mi formación han sido un apoyo importante. Espero que este logro sea motivo de orgullo para ustedes.

Agradezco a mis amigos: Oscar, Luz, Karla y Rony. Gracias por su compañía durante estos años, por todas las conversaciones, risas y su apoyo.

Finalmente, agradezco también al Dr. Luis Rivera por su acompañamiento y guía durante este trabajo de graduación.

Índice general

Índice de figuras

Índice de cuadros

Resumen

Este trabajo presenta el diseño y evaluación de un sistema basado en aprendizaje profundo para clasificar ventanas de EEG multi-paciente de la base Temple University Hospital EEG Corpus (TUSZ) como ictales o no ictales. Se implementó un *pipeline* en Python que incluye selección de canales, montaje, filtrado, remuestreo, segmentación en ventanas, balanceo de clases y extracción de características estadísticas y espectrales. Sobre este flujo se entrenaron tres arquitecturas: una red convolucional temporal (TCN), un modelo híbrido CNN+RNN con atención y un Transformer temporal, utilizando TensorFlow y PyTorch, con particiones entre pacientes y métricas estándar de clasificación.

Los resultados muestran que TCN e híbrida alcanzan el mejor compromiso entre desempeño y costo computacional en un escenario fuertemente desbalanceado, mientras que el Transformer presenta mayor complejidad y menor precisión efectiva. Se cuantificó además el impacto del hardware (PC de laboratorio, HPC e instancia en la nube) y de las optimizaciones del *pipeline* sobre los tiempos de entrenamiento. Finalmente, los modelos se integraron en la herramienta existente mediante una API y clientes en MATLAB y Python, habilitando la solicitud de predicciones sobre registros EDF y su visualización conjunta con anotaciones clínicas.

Palabras clave: epilepsia, EEG, aprendizaje profundo, TUSZ, TCN, CNN-RNN, Transformer, detección de crisis, cómputo de alto rendimiento, API.

Abstract

The following work presents the design and evaluation of a deep learning–based system to classify multi-patient EEG windows from the Temple University Hospital EEG Corpus (TUSZ) as ictal or non-ictal. A Python *pipeline* was implemented, including channel selection, montage, filtering, resampling, window segmentation, class balancing, and extraction of statistical and spectral features. On top of this flow, three architectures were trained: a Temporal Convolutional Network (TCN), a hybrid CNN+RNN model with attention, and a temporal Transformer, using TensorFlow and PyTorch, with cross-patient partitions and standard classification metrics.

The results show that the TCN and the hybrid model achieve the best trade-off between performance and computational cost in a highly imbalanced scenario, while the Transformer exhibits higher complexity and lower effective accuracy. The impact of hardware (laboratory PC, HPC cluster, and cloud instance) and pipeline optimizations on training time was also quantified. Finally, the models were integrated into the existing tool through an API and MATLAB and Python clients, enabling prediction requests on EDF recordings and their joint visualization with clinical annotations.

Keywords: epilepsy, EEG, deep learning, TUSZ, TCN, CNN-RNN, Transformer, seizure detection, high-performance computing, API.

CAPÍTULO 1

Introducción

La epilepsia es un trastorno neurológico crónico cuya evaluación clínica depende en gran medida del análisis de registros de electroencefalografía (EEG). Estos estudios capturan la actividad bioeléctrica cerebral en múltiples canales y generan grandes volúmenes de datos que deben ser revisados por especialistas para localizar episodios de convulsión y segmentos de interés, en un proceso demandante en tiempo y difícil de escalar. Esto hace necesario contar con herramientas computacionales que apoyen el análisis, faciliten la exploración de los registros y permitan experimentar con métodos de detección automática.

En la Universidad del Valle de Guatemala se ha desarrollado, a lo largo de varias fases, una línea de investigación orientada al estudio de la epilepsia mediante EEG y al diseño de una herramienta de software para su análisis. Este trabajo corresponde a una etapa de actualización de esa línea y se centra en el uso de técnicas de aprendizaje profundo para identificar segmentos de interés usando registros EEG de la base Temple University Hospital EEG Corpus (TUSZ). Para ello se diseña un *pipeline* que incluye selección de canales, montaje, filtrado, segmentación en ventanas, balanceo de clases y extracción de características, sobre el cual se implementan y comparan tres arquitecturas profundas (TCN, Transformer e híbrida), evaluando tanto su capacidad de detección como su eficiencia computacional en distintos entornos de hardware.

Finalmente, los modelos y *pipelines* desarrollados se integran a la herramienta existente mediante una arquitectura de servicios que expone el análisis a través de una API y lo conecta con interfaces en MATLAB y Python. De esta forma, los resultados trascienden el ámbito de prototipo y constituyen el primer paso hacia una plataforma extensible, apta para nuevos estudios, la exploración de configuraciones adicionales y, en el futuro, la evaluación de estas técnicas como apoyo al análisis clínico de registros EEG en pacientes con epilepsia.

CAPÍTULO 2

Antecedentes

2.1. La epilepsia

La epilepsia es una enfermedad neurológica caracterizada por una predisposición duradera a generar crisis epilépticas, que se diagnostica cuando una persona presenta crisis no provocadas y recurrentes, de acuerdo con los criterios propuestos por la ILAE [1]. Esta enfermedad tiene un alto impacto en la medicina, ya que provoca una condición debilitante en quienes la padecen. Además, representa un reto y un campo de avance en la medicina, puesto que dicho progreso ha contribuido a desestigmatizar una condición que, históricamente, fue motivo de discriminación [2].

Actualmente, la epilepsia afecta a alrededor de 50 millones de personas en el mundo, según la Organización Mundial de la Salud (OMS), de las cuales aproximadamente el 80 % residen en países en desarrollo [3]. Las personas con esta condición tienen un riesgo tres veces mayor de sufrir muertes prematuras y, en dichos países, tres cuartas partes de los afectados no reciben el tratamiento adecuado debido a dificultades económicas o a la discriminación. Se estima que, si se diagnostica y trata oportunamente, el 70 % de las personas podrían vivir sin convulsiones subsecuentes [4], [5]. Este panorama resalta la necesidad de contar con métodos de diagnóstico eficientes y precisos que permitan un mejor entendimiento de la enfermedad.

2.2. Aprendizaje profundo aplicado a la epilepsia

En los últimos años se han explorado varias maneras de utilizar métodos de aprendizaje automático (*Machine Learning* – ML) y aprendizaje profundo (*Deep Learning* – DL) en el análisis de señales bioeléctricas. Estos se han empleado con el fin de facilitar y automatizar el proceso de detección de convulsiones de manera precisa para el diagnóstico de la epilepsia. A continuación se presentan resultados obtenidos en diversos estudios con enfoque en la detección de convulsiones en pacientes epilépticos.

Cuadro 1. Resultados obtenidos por distintas arquitecturas de aprendizaje profundo en estudios de detección de convulsiones en pacientes epilépticos [6], [7], [8]

Arquitectura	Precisión (%)
RNN 2 capas LSTM y 1 capa densa	87
CNN 2 capas convolucionales y 1 capa densa	88
DBN 3 capas RBM y 1 capa densa	89
Combinación RNN y CNN	94
Combinación DBN y RNN	97

Nota. La tabla presenta un resumen comparativo de precisiones reportadas en la literatura para diferentes arquitecturas de aprendizaje profundo aplicadas a la detección de convulsiones en pacientes con epilepsia. Elaboración propia a partir de [6], [7], [8].

2.3. Línea de investigación en la Universidad del Valle

El trabajo de Angulo [9] y Pineda [10], bajo la supervisión del profesor Luis Alberto Rivera y en colaboración con el Centro de Epilepsia y Neurocirugía Funcional (*HUMANA*), inició el desarrollo de herramientas basadas en aprendizaje automático para el diagnóstico de la epilepsia a partir de señales de electroencefalografía (EEG). En este primer proyecto, Pineda implementó una base de datos en *MySQL* integrada en *MATLAB*, mientras que Angulo aplicó algoritmos de aprendizaje supervisado (ANN y SVM), alcanzando precisiones superiores al 95 % en la identificación de segmentos de interés.

En 2021, Jorge Manrique [11] optimizó la herramienta para un uso más eficiente en *HUMANA*, y David Vela [12] utilizó una SVM con kernel gaussiano para clasificar EEG en cuatro categorías (ictal, sano, preictal e interictal), obteniendo precisiones de 77.1 %, 97.2 % y 97.7 % en escenarios de cuatro, tres y dos clases, respectivamente.

La línea de investigación se expandió en 2022 con Camila Lemus [13] adoptando un enfoque no supervisado que combinó EEG y electrocardiogramas (ECG) para validar resultados previos de ANN, alcanzando una precisión de 99 % ($\pm$0.61 %) y reduciendo tiempos de entrenamiento mediante agrupamiento con *k-means* (85.51 % de precisión). Paralelamente, Samuel Silvestre [14] hizo público el repositorio de EEG, facilitando el acceso a los datos.

En 2023, Katherine Caceros [15] amplió y enriqueció el repositorio con nuevas señales biomédicas, Diego Méndez [16] migró la herramienta de *MATLAB* a un ejecutable independiente, y Christofer Patzán [17] aplicó técnicas basadas en wavelets junto con el *Chameleon Swarm Algorithm* para el análisis de EEG y electromiogramas (EMG).

En la última fase realizada en 2024 por Javier Pérez [18] y Dylan Ixcayau [19], se abordó el aprendizaje no supervisado y supervisado para etiquetar y clasificar señales EEG. Ixcayau aplicó métodos de agrupación *k-means*, agrupamiento jerárquico y *Fuzzy C-Means* validados con *Rand Index*, y combinó esta estrategia con PCA para preprocesamiento y normalización *z-score* para reducir el sesgo por varianza. Estos enfoques permitieron obtener mejores resultados en pruebas intrasujeto, mientras que, en conjuntos de datos extensos, los métodos de agrupamiento ofrecieron mejores resultados a costa de un mayor consumo de recursos computacionales.

Por otro lado, Pérez se centró en técnicas supervisadas empleando redes neuronales recurrentes (RNN) y redes neuronales convolucionales (CNN). En un primer enfoque con una red LSTM, a pesar de variaciones en la duración del entrenamiento (entre 2 y 35 horas) y en la cantidad de unidades ocultas, se alcanzó la mayor precisión (81.57 %) con configuraciones de 200-150 y 256-128 unidades. Posteriormente, mediante la implementación de una *Temporal Convolutional Network* (TCN) y variando parámetros como la duración del entrenamiento, bloques residuales, filtros y configuraciones de *kernel*, se obtuvo el mejor desempeño con el *kernel* más pequeño y el tiempo de entrenamiento más corto, sin superar una precisión del 81.4 %. El alcance del trabajo no permitió la exploración completa del uso de mayores recursos computacionales para entrenar con una mayor cantidad de datos.

CAPÍTULO 3

Justificación

En Guatemala, se estima que alrededor de 325,000 personas padecen epilepsia, y aproximadamente el 30 % de estos casos son de difícil control [2]. La limitada disponibilidad de servicios médicos adecuados en el país hace que el diagnóstico y tratamiento sean costosos, lo que añade una carga económica significativa. Ante este escenario, se ha vuelto esencial impulsar la investigación y el desarrollo de herramientas diagnósticas y terapéuticas, con el fin de optimizar el uso de los recursos en el contexto económico guatemalteco y mejorar la calidad de vida de los afectados, reduciendo al mismo tiempo el estigma asociado a la enfermedad.

El desarrollo de la *Epileptic EEG Analysis Toolbox* ha permitido avances significativos en la optimización y precisión de los métodos de aprendizaje automático empleados. La última fase basada en aprendizaje profundo evidenció algunas limitaciones, lo que abrió la puerta al uso de nuevos algoritmos y a la mejora de los existentes, así como al aprovechamiento de mayores recursos computacionales y al uso de diversos enfoques para evaluar el aumento de la precisión en la detección de segmentos de interés en señales bioeléctricas asociadas a la epilepsia.

En este proyecto se avanzó mediante la migración de métodos explorados anteriormente de MATLAB a Python, aprovechando la mayor accesibilidad a recursos computacionales y la amplia oferta de bibliotecas de software que este entorno ofrece. Esta migración no solo permitió evaluar si había mejoría en el rendimiento, sino que también posibilitó el análisis de conjuntos de datos más extensos, aprovechando la creciente cantidad de datos obtenidos en investigaciones previas.

La optimización y migración de los métodos utilizados en fases anteriores buscaron mejorar la precisión en la detección de segmentos de interés. Además, trasladarlos a Python facilitó su entrenamiento y actualización, ya que Python y sus bibliotecas de software son de acceso gratuito y cuentan con un robusto soporte, lo que simplifica la integración de *APIs* y la creación de ejecutables que complementen el uso de MATLAB al facilitar el acceso a los datos obtenidos con los algoritmos. También facilitó la utilización del hardware de computadoras de alto rendimiento, con el objetivo de evaluar si un conjunto mayor de datos de

entrenamiento podría mejorar la precisión.

Finalmente, este trabajo se justifica también por su contribución a la consolidación de una línea de investigación local en epilepsia y señales bioeléctricas, al proveer modelos reproducibles y una infraestructura más accesible para futuros proyectos. Esto favorece la transferencia de conocimiento hacia otros grupos de investigación y abre la posibilidad de una integración progresiva de estas herramientas en el apoyo al diagnóstico clínico en centros especializados como HUMANA.

CAPÍTULO 4

Objetivos

4.1. Objetivo general

Aplicar algoritmos de aprendizaje profundo para la identificación de segmentos de interés en señales bioeléctricas relacionadas con la epilepsia.

4.2. Objetivos específicos

- Evaluar herramientas y librerías en Python para implementar y optimizar métodos de aprendizaje profundo, seleccionando las arquitecturas neuronales y los algoritmos más adecuados para la detección de segmentos de interés en señales EEG correspondientes a la epilepsia.
- Aplicar los algoritmos de aprendizaje profundo seleccionados a una cantidad substancial de señales bioeléctricas realcionadas a estudios de epilepsia en la búsqueda de segmentos de interés.
- Evaluar los resultados de los algoritmos utilizando métodos estadísticos.
- Explorar recursos computacionales de alto rendimiento para ejecutar los algoritmos de aprendizaje profundo seleccionados y comparar sus ventajas en tiempo de cómputo y efectividad respecto a recursos convencionales.
- Actualizar la herramienta de software para el estudio de la epilepsia desarrollada en fases anteriores, incorporando los nuevos algoritmos y las mejoras realizadas en esta fase.

CAPÍTULO 5

Definición del problema

Se diseñó, implementó y evaluó un conjunto de métodos de aprendizaje profundo para identificar segmentos de interés en EEG asociados a epilepsia, continuando la línea de investigación de la UVG. El trabajo se apoyó en desarrollos previos y concretó la migración de métodos desde MATLAB a Python para aprovechar bibliotecas de software especializadas, evaluar métricas de aprendizaje y velocidad de entrenamiento mediante procedimientos estadísticos establecidos y agilizar el entrenamiento. Con ello se mejoró la accesibilidad, la escalabilidad y el mantenimiento de las arquitecturas y del flujo de procesamiento de archivos EDF.

El alcance comprendió la puesta a punto del entorno y de los modelos en Python, junto con la preparación y organización de datos EEG a partir de archivos EDF. Se implementaron y probaron tres arquitecturas: una *Temporal Convolutional Network* (TCN) derivada de la fase anterior, una arquitectura híbrida con capas convolucionales, recurrentes y atención, y un modelo *Transformer* con parámetros ajustables para analizar su impacto en el entrenamiento. Todo quedó integrado en un código modular que permitió modificar tanto las arquitecturas como el flujo de procesamiento y la generación de conjuntos de datos. Los resultados se incorporaron en la herramienta existente y se realizó una comparación del desempeño computacional en condiciones controladas. No se abordó validación clínica ni operación en tiempo real, por lo que el uso se mantuvo en el ámbito de investigación. La migración quedó documentada y operativa, con capacidad de entrenar y actualizar modelos y de ejecutar análisis sobre volúmenes ampliados de registros.

CAPÍTULO 6

Marco teórico

6.1. La epilepsia

La epilepsia es una afección crónica que se manifiesta en episodios recurrentes. Durante una crisis, las neuronas disparan impulsos eléctricos de forma desordenada y excesiva, alterando el funcionamiento normal del cerebro y llegando a provocar cambios bruscos en el comportamiento [2]. Esta enfermedad afecta alrededor del 1 % de la población mundial, siendo más prevalente y de difícil control en países en vías de desarrollo. En Guatemala, aproximadamente 350,000 personas padecen epilepsia y alrededor del 30 % de los casos son de difícil control [20].

La epilepsia agrupa diferentes trastornos neurológicos que se distinguen según la edad de aparición y la zona del cerebro afectada. Sus crisis pueden durar distintos lapsos y manifestarse de formas muy variadas: desde alucinaciones o alteraciones cognitivas hasta contracciones musculares focales o generalizadas, que en ocasiones no implican pérdida de conciencia. En la práctica clínica, el diagnóstico suele confirmarse cuando se han presentado al menos dos de estos episodios no provocados y recurrentes [2], [21].

6.1.1. Categorización de crisis epilépticas

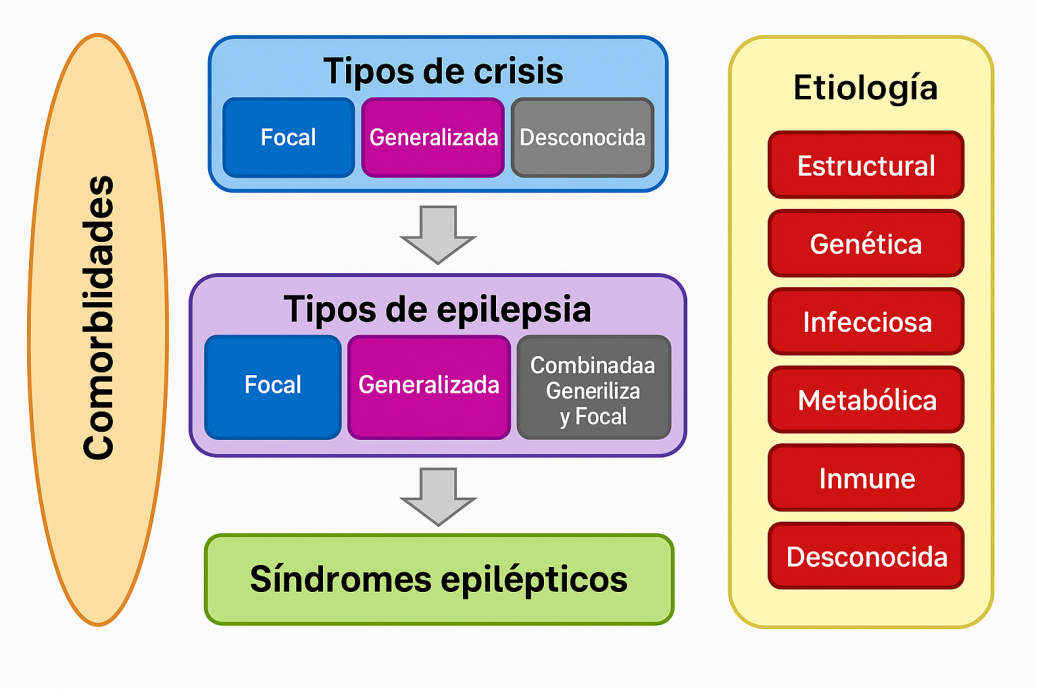
Según la última revisión de la Liga Internacional Contra la Epilepsia (*ILAE, International League Against Epilepsy*), las crisis epilépticas se clasifican según tres criterios: el impacto motor, el nivel de conciencia y el origen cerebral.

- Inicio focal: suceden desde una región específica del cerebro y se clasifican primero por el nivel de conciencia (conscientes o con conciencia alterada), luego por su manifestación motora (tónicas, clónicas o automatismos) o no motora (sensoriales, cognitivas o autonómicas) y, en algunos casos, por su evolución al expandirse a otras áreas (bilateral tónico-clónica) [22], [23].

- Inicio generalizado: suceden cuando la actividad eléctrica se dispara a la vez en ambos hemisferios cerebrales, provocando al mismo tiempo síntomas motores y alteraciones de la conciencia. Su forma más conocida son las convulsiones tónico-clónicas, que combinan una fase de rigidez con sacudidas musculares rítmicas y casi siempre pérdida de conciencia, y también las crisis de ausencia, breves lapsos de desconexión en los que apenas se advierten movimientos [22], [23].
- Inicio desconocido: cuando no se puede identificar con certeza si la crisis comienza de forma focal o generalizada, se considera como de inicio desconocido. La forma de clasificarla es según los síntomas motores que presente y si implica o no alteración de la conciencia [22], [23].

La Figura 1 resume esta clasificación operacional y su relación con los tipos de epilepsia, síndromes, comorbilidades y causas etiológicas.

Figura 1. Clasificación operacional de la epilepsia según la ILAE 2017



Nota. La figura muestra la relación entre los tipos de crisis, los tipos de epilepsia, los síndromes epilépticos, las comorbilidades y las causas etiológicas según la clasificación propuesta por la Liga Internacional Contra la Epilepsia (ILAE). Elaboración propia basada en la guía presentada por Fisher et al. (2017) [22].

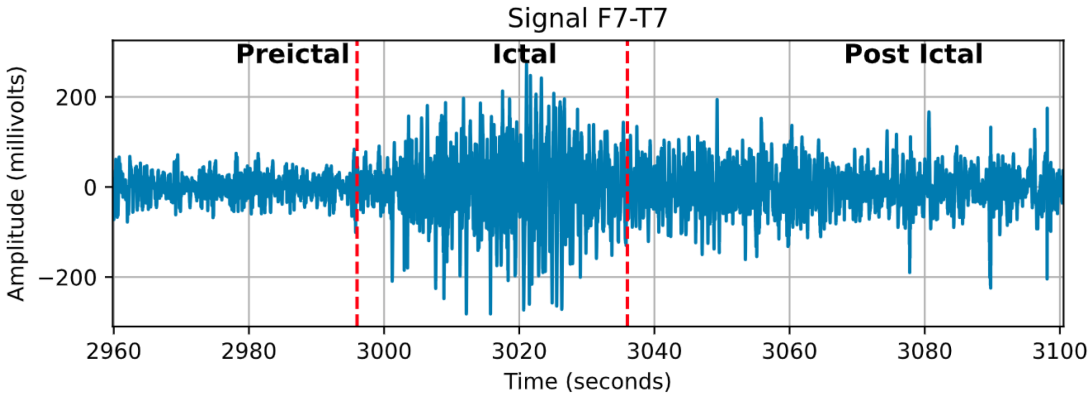
6.1.2. Fases de crisis epilépticas

Las crisis epilépticas transcurren en fases, cada una con síntomas y señales propias. Conocerlas y saber identificarlas ayuda a entender cómo progresa la crisis e incluso a detectarla antes de que se desarrolle por completo.

- Fase preictal: es la fase que precede a la crisis, marcada por cambios sutiles en la actividad cerebral que aún no son inmediatamente evidentes. Durante este lapso, que puede durar desde unos pocos segundos hasta cerca de una hora, aparecen síntomas cognitivos como sensación de fatiga, distorsiones sensoriales o desconexión de la realidad, junto a movimientos involuntarios leves, como espasmos musculares o sacudidas breves. Estos signos iniciales sirven de aviso temprano antes de que se manifieste la etapa más intensa de la crisis [2].
- Fase ictal: la fase ictal se caracteriza por los síntomas más notorios. Aparecen contracciones o movimientos musculares involuntarios, prolongados o rítmicos, que pueden afectar una o varias regiones del cuerpo y suelen ir acompañados de alteraciones cognitivas. Estos signos se producen por descargas eléctricas cerebrales desorganizadas con patrones distintivos que provocan los síntomas desde unos segundos hasta varios minutos, sin que necesariamente se presenten convulsiones [2].
- Fase postictal: esta fase, posterior a la fase ictal, suele manifestarse con confusión, debilidad o parálisis temporal de una o varias extremidades y una sensación de agotamiento intenso. Estos efectos pueden prolongarse desde varias horas hasta incluso días, con variaciones en su duración y gravedad según cada persona [2].

La Figura 2 ilustra un ejemplo de registro EEG en el que se señalan estas fases sobre una misma señal.

Figura 2. Ejemplo de señal EEG segmentada en fases preictal, ictal y postictal



Nota. La figura muestra un ejemplo de registro EEG del canal F7–T7, en el cual se identifican las fases preictal, ictal y postictal mediante el análisis de la variación de la amplitud de la señal. El eje horizontal (*Time (seconds)*) representa el tiempo en segundos, mientras que el eje vertical (*Amplitude (millivolts)*) indica la amplitud del voltaje en milivoltios. Imagen obtenida de [24].

6.2. Señales bioeléctricas

Son señales obtenidas de la actividad eléctrica de neuronas o células musculares, se registran de manera continua a lo largo del tiempo y permiten identificar características propias de su origen en aplicaciones médicas [25]. En este trabajo se hace énfasis en los electroencefalogramas (EEG), que miden la actividad cerebral.

6.2.1. Electroencefalogramas

Los electroencefalogramas (EEG) registran la actividad eléctrica del cerebro a lo largo del tiempo, traduciendo las variaciones de amplitud en información sobre su funcionamiento. Para ello, se usan montajes que sitúan electrodos en puntos precisos del cuero cabelludo, de modo que captan las señales generadas por las neuronas [26]. En el contexto de la epilepsia, los EEG resultan esenciales para aislar los segmentos de interés, facilitar el diagnóstico, clasificar el tipo de crisis y relacionarlas con síndromes específicos.

6.2.2. Análisis en el dominio del tiempo

El análisis en el dominio del tiempo parte de la señal EEG sin aplicar transformaciones previas y se centra en extraer métricas que describan su comportamiento y evolución. Entre las medidas más empleadas están la media, la varianza, la desviación estándar, la asimetría, la curtosis y el número de cruces por cero. Para visualizar estos cambios a lo largo del tiempo se usan histogramas y trazados de tendencia, que permiten observar la dinámica de amplitudes en distintas escalas temporales. Gracias a estas herramientas es posible detectar variaciones bruscas, patrones repetitivos o periodos de estabilidad que resultan clave tanto en entornos clínicos como en estudios de investigación [27].

6.2.3. Análisis en el dominio de la frecuencia

El análisis en el dominio de la frecuencia transforma la señal EEG desde el tiempo para obtener su espectro de frecuencias, normalmente a través de la densidad espectral de potencia (PSD) o su versión normalizada. A partir de ese espectro se extraen características como la potencia absoluta y relativa en bandas específicas, la frecuencia dominante o centroide y otros índices espectrales que condensan información compleja en unos pocos valores manejables. Estos índices resultan menos sensibles al ruido que afecta a los electrodos y pueden emplearse de forma individual o combinarse entre sí para definir nuevas variables de interés, facilitando el análisis comparado y la interpretación de la dinámica cerebral sin revisar toda la variación temporal de la señal [27].

6.2.4. Análisis en el dominio de tiempo–frecuencia

El análisis en el dominio tiempo–frecuencia permite describir señales no estacionarias mediante representaciones que combinan la evolución temporal y espectral de la actividad cerebral. En lugar de un espectro único promedio, se emplean herramientas como la transformada de Fourier de tiempo corto (STFT) o las transformadas wavelet para obtener mapas en los que la energía se distribuye en función del tiempo y la frecuencia [28]. A partir de estos mapas se derivan características como energía por banda, razones entre bandas o descriptores estadísticos que capturan cambios transitorios asociados a crisis epilépticas, y que pueden utilizarse tanto como entrada directa de modelos de aprendizaje profundo como para complementar las características en dominio del tiempo y de la frecuencia.

6.3. Dinámica y espacio de Koopman

La teoría de Koopman propone estudiar sistemas no lineales mediante la evolución de funciones del estado que se denominan observables a través de un operador lineal de dimensión en general infinita. En la práctica se aproxima con métodos como descomposición en modos dinámicos (DMD) y su extensión (EDMD). Estas técnicas descomponen la dinámica en autovalores vinculados a frecuencias y tasas de amortiguamiento y en modos espaciales que representan patrones sobre los canales [29], [30], [31].

6.4. Aprendizaje automático y profundo

6.4.1. Aprendizaje automático

El aprendizaje automático reúne métodos que aprenden regularidades a partir de ejemplos para reconocer patrones y realizar predicciones. El entrenamiento consiste en encontrar los valores de esos parámetros que minimizan una función de pérdida, la cual cuantifica la discrepancia entre las predicciones y los objetivos.

Según la disponibilidad de etiquetas se distinguen enfoques de aprendizaje supervisado, no supervisado y por refuerzo. El aprendizaje supervisado se basa en pares entrada–salida para tareas de clasificación o regresión. El aprendizaje no supervisado busca estructura en los datos sin etiquetas, por ejemplo mediante agrupamiento. El aprendizaje por refuerzo modela agentes que toman decisiones secuenciales para maximizar una recompensa acumulada. La minimización de la pérdida se resuelve con algoritmos de optimización iterativos y se evalúa con particiones de entrenamiento, validación y prueba para verificar la generalización.

En EEG se emplean representaciones que resaltan distintos aspectos de la señal. El dominio del tiempo describe morfología, duración y transitorios, mientras que los dominios de la frecuencia y del tiempo–frecuencia caracterizan ritmos y bandas asociadas a estados o eventos. Estas representaciones pueden ser la entrada del modelo o servir de referencia para interpretar modelos que operan sobre la señal cruda [28].

6.4.2. Redes neuronales

Una red neuronales un modelo compuesto por unidades simples, llamadas neuronas, organizadas en capas. Cada neurona recibe entradas, calcula una combinación lineal de ellas mediante pesos y un sesgo, y aplica una función de activación que introduce no linealidad. Las capas de entrada reciben los datos originales, las capas ocultas transforman progresivamente esas representaciones y la capa de salida produce la predicción final.

En su forma más simple, una red neuronal tiene una sola capa oculta y se utiliza para tareas de clasificación o regresión. Al ajustar los pesos y sesgos durante el entrenamiento, el modelo aprende a aproximar funciones que relacionan las entradas con las salidas. Además de la función de pérdida, se monitorizan métricas como exactitud y sensibilidad para juzgar el desempeño según las prioridades del problema. La anchura de las capas, la elección de las

funciones de activación y las técnicas de regularización influyen en la capacidad del modelo para generalizar a datos no vistos.

6.4.3. Aprendizaje profundo

El aprendizaje profundo se refiere al entrenamiento de redes neuronales con múltiples capas ocultas para aprender representaciones jerárquicas a partir de los datos. En las primeras capas se capturan rasgos locales y de baja complejidad, mientras que las capas posteriores combinan esas representaciones para modelar estructuras más abstractas y útiles para la tarea.

Estas redes pueden entrenarse en régimen supervisado cuando se dispone de etiquetas, en régimen no supervisado cuando se busca descubrir estructura sin etiquetas y mediante aprendizaje por refuerzo cuando se optimiza una secuencia de decisiones guiada por recompensas. Su rasgo distintivo es la capacidad de aprender directamente de grandes volúmenes de datos y de mejorar el desempeño al aumentar la cantidad y diversidad de ejemplos, siempre que se acompañe de técnicas adecuadas de regularización, optimización y evaluación [32], [33].

6.4.4. Funciones de pérdida

La función de pérdida mide la discrepancia entre la predicción del modelo y el objetivo. Su elección define qué errores se penalizan más y cómo se guía el aprendizaje.

Entropía cruzada

La entropía cruzada mide la discrepancia entre la distribución objetivo y y la distribución de probabilidades predicha $\hat{p}$. Para K clases se define como

$$\mathcal{L}_{\mathrm{CE}} = -\sum_{k=1}^{K} y_k \, \log\left(\hat{p}_k\right), \tag{1}$$

donde y asigna toda la probabilidad a la clase correcta y $\hat{p}$ reparte probabilidad entre las K clases de forma que sus componentes son no negativos y suman uno. En el caso binario, esta expresión se reduce a

$$\mathcal{L}_{\mathrm{CE}} = -\left[\, y \log(\hat{p}) + (1-y)\log(1-\hat{p}) \,\right], \tag{2}$$

donde $\hat{p}$ representa la probabilidad predicha de la clase positiva. Esta pérdida favorece modelos cuyas probabilidades se aproximan a las verdaderas, lo que se traduce en predicciones mejor calibradas cuando los datos son representativos [32], [33].

Pérdida focal

La pérdida focal modifica la entropía cruzada para restar peso a ejemplos fáciles y concentrar el aprendizaje en los difíciles:

$$\mathcal{L}_{\text{Focal}} = -\alpha\,(1-\hat{p})^{\gamma}\,y\,\log(\hat{p}) \;-\; (1-\alpha)\,\hat{p}^{\gamma}\,(1-y)\,\log(1-\hat{p}), \tag{3}$$

donde $\gamma > 0$ controla el énfasis y $\alpha \in [0,1]$ ayuda a manejar desbalances de clase. La idea se extiende a multiclase aplicando el mismo factor modulador a la probabilidad de la clase objetivo [34].

Pérdida de Tversky

La pérdida de Tversky deriva de un índice de similitud que permite ponderar de forma asimétrica los falsos positivos y los falsos negativos. Con verdaderos positivos TP, falsos positivos FP y falsos negativos FN, el índice de Tversky se define como

$$\text{TI} = \frac{\text{TP}}{\text{TP} + \alpha\,\text{FP} + \beta\,\text{FN}}, \qquad \mathcal{L}_{\text{Tversky}} = 1 - \text{TI}. \tag{4}$$

Los parámetros α y β controlan la penalización. Al fijar $\alpha = \beta = 0.5$ se obtiene el *coeficiente de Dice*, una medida ampliamente usada en segmentación:

$$\text{Dice} = \frac{2\,\text{TP}}{2\,\text{TP} + \text{FP} + \text{FN}}, \qquad \mathcal{L}_{\text{Dice}} = 1 - \text{Dice}. \tag{5}$$

Existe una relación con F_β, que equilibra precisión y sensibilidad mediante un factor β que da mayor peso a la sensibilidad cuando $\beta > 1$:

$$F_\beta = \frac{(1+\beta^2)\,\text{Prec}\,\text{Sens}}{\beta^2\,\text{Prec} + \text{Sens}}. \tag{6}$$

La familia de Tversky generaliza estas métricas al permitir un ajuste fino del compromiso entre falsos positivos y falsos negativos. En conjuntos desbalanceados resulta útil seleccionar $\beta > \alpha$ para priorizar la detección de la clase minoritaria y reducir omisiones críticas [35].

Regularización L_2

Además de la pérdida principal, se suele añadir un término de regularización que penaliza parámetros de gran magnitud. La regularización L_2 incorpora un término proporcional a la norma cuadrática de los parámetros,

$$\mathcal{L}_{\text{reg}}(\theta) = \lambda\,\|\theta\|_2^2, \tag{7}$$

donde $\lambda > 0$ controla la intensidad de la penalización. Este término desalienta pesos muy grandes, reduce el sobreajuste y está estrechamente relacionado con las técnicas de *weight decay* utilizadas en los algoritmos de optimización.

6.4.5. Descenso de gradiente y algoritmos de optimización

El descenso de gradiente y sus variantes son los procedimientos numéricos que se usan para entrenar modelos ajustando sus parámetros con el fin de minimizar la función de pérdida. Dado un conjunto de parámetros θ y una pérdida $\mathcal{L}(\theta)$ que mide el error de las predicciones frente a los objetivos, una actualización básica por descenso de gradiente en el paso t se escribe como

$$\theta_{t+1} = \theta_t - \eta \, \nabla_\theta \mathcal{L}(\theta_t), \tag{8}$$

donde η es la tasa de aprendizaje y $\nabla_\theta \mathcal{L}(\theta_t)$ es el gradiente de la pérdida con respecto a los parámetros en ese instante.

En aprendizaje profundo se emplea con frecuencia el esquema por mini–lotes o *batches*, que aproxima el gradiente usando pequeños subconjuntos de datos para ganar eficiencia y tolerar ruido. La tasa de aprendizaje controla el tamaño del paso y, junto con mecanismos como el momento y la normalización de gradientes, determina la estabilidad y la velocidad de convergencia. Debido a que la forma de la pérdida en redes profundas es no convexa, el objetivo práctico no es garantizar el mínimo global, sino alcanzar soluciones de bajo error que generalicen bien a datos no vistos [32], [33].

Adam

Adam combina promedios móviles exponenciales del gradiente y de su cuadrado, que corresponden a estimaciones adaptativas de primer y segundo momento. En el paso t, dado un gradiente g_t, las estimaciones se actualizan como

$$m_t = \beta_1 m_{t-1} + (1-\beta_1)\, g_t, \qquad v_t = \beta_2 v_{t-1} + (1-\beta_2)\, g_t^2, \tag{9}$$

donde m_t y v_t aproximan el primer y segundo momento del gradiente, y $\beta_1, \beta_2 \in [0, 1)$ controlan el decaimiento exponencial. Para corregir el sesgo inicial se usan

$$\hat{m}_t = \frac{m_t}{1-\beta_1^t}, \qquad \hat{v}_t = \frac{v_t}{1-\beta_2^t}, \tag{10}$$

y la actualización de parámetros toma la forma

$$\theta_{t+1} = \theta_t - \eta \, \frac{\hat{m}_t}{\sqrt{\hat{v}_t} + \epsilon}, \tag{11}$$

donde $\epsilon > 0$ evita divisiones por cero. Esta combinación ofrece robustez frente a gradientes ruidosos y magnitudes heterogéneas [36].

AdamW

AdamW desacopla el decaimiento por pesos de la actualización por gradiente. En lugar de integrar la penalización L_2 en la magnitud adaptativa del paso, aplica una reducción explícita de los parámetros en cada iteración. Una forma sencilla de expresar este desacoplamiento es

$$\theta_{t+1} = \theta_t - \eta \, \frac{\hat{m}_t}{\sqrt{\hat{v}_t} + \epsilon} - \eta \, \lambda \, \theta_t, \tag{12}$$

donde el último término implementa el *weight decay* con factor λ. Esta separación preserva el efecto regularizador del decaimiento de pesos sin interferir con la adaptación de escalas, lo que mejora la generalización en diversos escenarios [37].

Momento y promedio móvil exponencial

El momento clásico acumula una fracción del gradiente previo para mantener inercia en direcciones consistentes y amortiguar el ruido. El promedio móvil exponencial asigna mayor peso a gradientes recientes que a los antiguos y controla su "memoria" mediante un factor de decaimiento. Adam emplea promedios móviles exponenciales tanto del gradiente como de su segundo momento. Además, es común mantener un promedio móvil exponencial de los parámetros para evaluar un modelo suavizado al final del entrenamiento [32], [36].

6.4.6. Funciones de activación

Las funciones de activación son los componentes que introducen no linealidad en las redes profundas. Gracias a ellas, las capas pueden combinar señales de forma flexible y representar relaciones complejas que una transformación lineal no puede capturar. Durante el entrenamiento, la activación elegida influye en la estabilidad del gradiente, en la velocidad de convergencia y en la capacidad de generalización del modelo [32].

ReLU

La unidad lineal rectificada se define como

$$\mathrm{ReLU}(x) = \mathrm{máx}(0, x). \tag{13}$$

Anula las activaciones negativas y deja pasar sin cambios las positivas. Esta forma simple acelera el entrenamiento, evita saturaciones frecuentes en funciones sigmoides y facilita la propagación del gradiente en capas profundas. Su principal debilidad aparece cuando muchas unidades quedan en la región negativa durante varias iteraciones. En ese caso, el gradiente se vuelve cero y esas unidades dejan de actualizarse, fenómeno conocido como "neurona muerta". Aun con esta limitación, ReLU se mantiene como una elección sólida en arquitecturas modernas por su eficiencia computacional y su comportamiento estable [32].

GELU

La unidad lineal con error gaussiano pondera la entrada por su probabilidad de ser "útil" según una distribución normal estándar. Se define como

$$\mathrm{GELU}(x) \;=\; x\,\Phi(x) \;=\; \tfrac{1}{2}\,x\left(1+\mathrm{erf}\left(\frac{x}{\sqrt{2}}\right)\right), \tag{14}$$

donde $\Phi(x)$ es la función de distribución normal. A diferencia de ReLU, que corta de forma abrupta en cero, GELU atenúa de manera suave las entradas negativas y conserva en mayor

medida las pequeñas positivas. Esta transición continua tiende a mejorar la optimización en modelos con gran profundidad y ha mostrado buenos resultados en transformadores y redes de lenguaje. En implementación práctica suele emplearse una aproximación eficiente basada en funciones hiperbólicas para reducir el costo de cómputo sin perder precisión [38].

6.4.7. Redes neuronales convolucionales (CNN)

Las redes neuronales convolucionales usan filtros que se desplazan sobre la entrada y generan mapas de activación donde aparecen los patrones de interés. La compartición de parámetros reduce el número de pesos y aporta equivarianza traslacional. Tras cada convolución se aplican no linealidades y, cuando conviene, normalización y técnicas de reducción de resolución para ganar robustez. Al apilar varias capas, la red pasa de rasgos locales a representaciones más abstractas y de mayor escala [32].

En EEG, las CNN trabajan de dos formas principales. La primera opera en una dimensión directamente sobre la señal temporal para captar transitorios y ritmos por canal y entre canales. La segunda utiliza representaciones tiempo–frecuencia y convoluciones bidimensionales para modelar conjuntamente variaciones en tiempo y banda. Ambas estrategias son complementarias y pueden integrarse en una misma arquitectura [28], [39].

6.4.8. Redes recurrentes (RNN): LSTM y GRU

Las redes recurrentes procesan secuencias paso a paso y mantienen un estado interno que resume lo visto hasta el momento. Este estado se actualiza con cada nueva entrada y permite representar dependencias temporales más allá de la ventana inmediata [32].

LSTM. La Long Short-Term Memory introduce una *celda* con rutas de información controladas por compuertas. La compuerta de olvido decide qué parte del estado previo se conserva, la compuerta de entrada regula cuánta información nueva se añade y la compuerta de salida determina qué porción del estado se expone como activación. Esta organización mitiga gradientes que se desvanecen y favorece dependencias de mayor alcance [40].

GRU. La Gated Recurrent Unit simplifica la estructura al combinar estado y salida en un solo vector y al utilizar dos compuertas. La compuerta de actualización mezcla información pasada y nueva, mientras que la compuerta de reinicio controla cuánta memoria previa se ignora al proponer el estado candidato. GRU mantiene buena capacidad de memoria con menor costo que LSTM y suele entrenar con mayor rapidez [41].

6.4.9. Atención

La atención es un mecanismo que pondera las partes de una secuencia según su relevancia para la tarea. Dado un vector de consulta, el modelo calcula similitudes con un conjunto de claves y usa esos pesos para promediar los valores asociados. El resultado es una combinación contextual que concentra la información útil y reduce la interferencia de elementos irrelevantes.

En su forma *multi-cabeza*, el mecanismo de atención se descompone en varias "cabezas" que operan en subespacios distintos de la representación. Cada cabeza aprende patrones complementarios y sus salidas se concatenan para formar una representación enriquecida [32], [42].

6.4.10. Transformers

Los *transformers* construyen representaciones de secuencias utilizando bloques de *autoatención* y capas *feed-forward*, sin recurrencia explícita. Cada elemento de la secuencia se relaciona con todos los demás mediante múltiples cabezas de atención que capturan patrones a distintas escalas. Para conservar el orden se añaden codificaciones posicionales a las entradas. Esta organización paraleliza el cómputo, facilita el entrenamiento en grandes conjuntos de datos y ha logrado resultados de referencia en diversas tareas de secuencias.

En señales como EEG, los transformers permiten modelar dependencias de largo alcance y asignar pesos diferenciados a segmentos informativos, con la posibilidad de ajustar el costo mediante ventanas locales, atenciones dispersas o reducciones de resolución temporal [28], [43].

6.4.11. Redes híbridas

Las redes híbridas combinan bloques de distinta naturaleza para aprovechar sus fortalezas en una misma arquitectura. Un diseño habitual integra capas convolucionales para extraer patrones locales, capas recurrentes para modelar dependencias a largo plazo y mecanismos de atención para resaltar la información más relevante.

En señales como EEG, este enfoque permite detectar transitorios en ventanas cortas, seguir su evolución en el tiempo y asignar mayor peso a segmentos que aportan evidencia a la decisión. La modularidad facilita ajustar cada bloque al problema y al presupuesto computacional [32], [39].

6.5. Métricas de evaluación utilizadas para determinar el rendimiento de Redes Neuronales

En un problema de clasificación binaria se consideran cuatro tipos de resultado: verdaderos positivos (VP), verdaderos negativos (VN), falsos positivos (FP) y falsos negativos (FN). A partir de estos valores se definen las siguientes métricas de desempeño [32], [33]:

Exactitud (*accuracy*). Proporción de aciertos totales:

$$\text{Exactitud} = \frac{VP + VN}{VP + VN + FP + FN}.$$

Resume el desempeño global, pero puede ser engañosa con clases desbalanceadas [33].

Pérdida (*loss*). Función objetivo que el modelo minimiza durante el entrenamiento (por ejemplo, entropía cruzada). No es una métrica de clasificación per se, pero indica la calidad del ajuste: pérdidas menores suelen asociarse a mejores probabilidades calibradas [32].

Sensibilidad (recuperación o *recall* de la clase positiva). Capacidad para detectar eventos positivos (convulsión):

$$\text{Sensibilidad} = \frac{VP}{VP + FN}.$$

Alta sensibilidad implica pocos falsos negativos [33].

Especificidad. Capacidad para reconocer correctamente los negativos (no convulsión):

$$\text{Especificidad} = \frac{VN}{VN + FP}.$$

Alta especificidad implica pocos falsos positivos [33].

F1 *score*. Media armónica entre precisión y recuperación de la clase positiva:

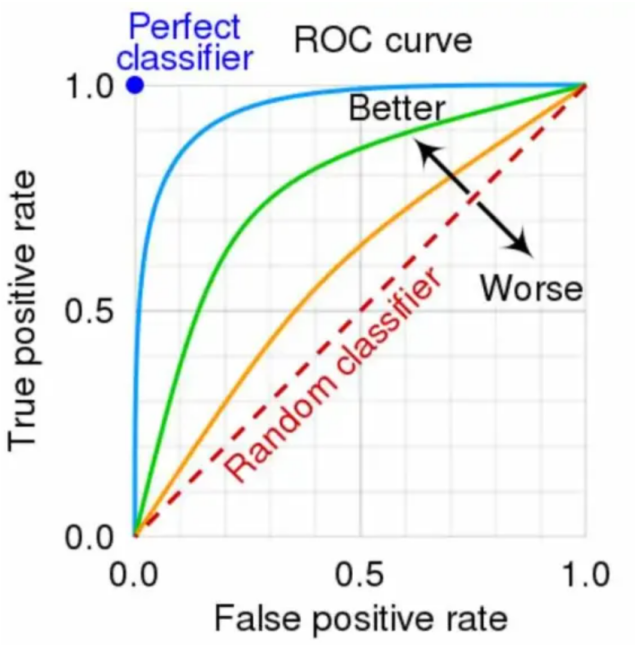
$$\text{F1} = 2 \cdot \frac{\text{Precisión} \cdot \text{Recuperación}}{\text{Precisión} + \text{Recuperación}}, \quad \text{donde} \quad \text{Precisión} = \frac{VP}{VP + FP}.$$

Es útil cuando interesa equilibrar los errores FP y FN y el conjunto está desbalanceado [32], [33].

Área bajo la curva PR (PR AUC). Área bajo la curva *Precisión–Recuperación* al variar el umbral de decisión, como se ilustra en la Figura 4. Resume el compromiso entre precisión y sensibilidad centrado en la clase positiva. Es especialmente informativa cuando la clase positiva es rara, como se ha destacado en el análisis de clasificadores sobre conjuntos desbalanceados [44], [45].

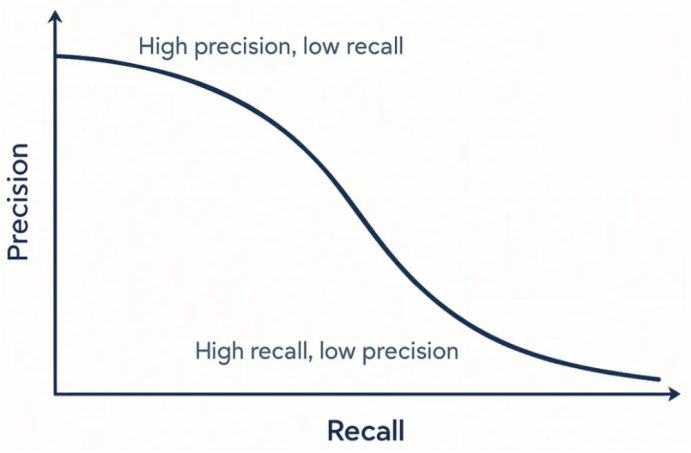
Área bajo la curva ROC (AUC ROC). Área bajo la curva *VPF* vs. *FPF* (tasa de verdaderos positivos frente a tasa de falsos positivos) al variar el umbral, como se muestra en la Figura 3. Mide la capacidad de ranquear correctamente positivos por encima de negativos. Valores cercanos a 1 indican excelente separabilidad, mientras que 0.5 equivale a azar [46], [47].

Figura 3. Ejemplo ilustrativo de curvas ROC para clasificadores con distinto desempeño



Nota. La figura muestra curvas ROC (*Receiver Operating Characteristic*) para clasificadores con diferente calidad. El eje horizontal corresponde a la tasa de falsos positivos (*False Positive Rate*) y el eje vertical a la tasa de verdaderos positivos (*True Positive Rate*). El punto etiquetado como *Perfect classifier* representa un clasificador ideal; la línea punteada indica un clasificador aleatorio (*Random classifier*); las curvas más cercanas a la esquina superior izquierda se consideran mejores (*Better*) y las próximas a la diagonal, peores (*Worse*). Imagen obtenida de [47].

Figura 4. Ejemplo ilustrativo de la curva Precisión–*Recall*



Nota. La figura muestra el compromiso entre precisión (*Precision*, eje vertical) y sensibilidad o recuperación (*Recall*, eje horizontal). La región próxima a la esquina superior izquierda corresponde a alta precisión y baja *recall* (*High precision, low recall*), mientras que la parte inferior derecha indica alta *recall* y baja precisión (*High recall, low precision*). Este tipo de curva es útil para analizar clasificadores en conjuntos desbalanceados. Imagen obtenida de [45].

6.6. Recursos computacionales para el entrenamiento de redes profundas

El entrenamiento de redes profundas demanda una cantidad considerable de cómputo y memoria. Cada iteración requiere evaluar el modelo sobre lotes de datos, calcular gradientes y actualizar millones de parámetros. El costo crece con el tamaño del conjunto de datos, la profundidad de la red y la resolución de las entradas. Para sostener ciclos de experimentación razonables, se combinan dos estrategias: acelerar la aritmética de tensores y mantener un flujo de datos constante desde el almacenamiento hasta el dispositivo de cómputo. En la práctica, esto implica usar aceleradores especializados, optimizar la precisión numérica y paralelizar el trabajo cuando el modelo o los datos superan la capacidad de un único dispositivo [32], [33].

6.6.1. Importancia de las GPU y del paralelismo

Las Unidades de Procesamiento Gráfico (GPU) concentran miles de núcleos simples capaces de ejecutar operaciones idénticas sobre grandes bloques de datos. Esta organización es ideal para las operaciones básicas del aprendizaje profundo, como las convoluciones y las multiplicaciones de matrices. El desarrollo de GPU impulsó saltos de rendimiento y posibilitó modelos más profundos y grandes conjuntos de datos. [39], [48]

El aumento de rendimiento se apoya en tres pilares. Primero, el paralelismo masivo permite computar en simultáneo miles de productos y sumas. Segundo, bibliotecas altamente optimizadas como cuDNN y cuBLAS ajustan detalles de bajo nivel y aprovechan instrucciones especializadas para tensores [49]. Tercero, la precisión mixta reduce el costo al representar activaciones y gradientes con formatos de menor tamaño manteniendo acumulaciones en 32 bits para preservar la estabilidad [50]. Al combinar estos factores, una sola GPU moderna puede acelerar órdenes de magnitud respecto a CPU generales.

Cuando una GPU deja de ser suficiente por tamaño de modelo o por volumen de datos, se recurre a entrenamiento distribuido. El paralelismo de datos replica el modelo en varias GPU y reparte los lotes. El paralelismo de modelo divide capas o matrices entre dispositivos. Bibliotecas de comunicación de alto rendimiento, como NCCL, sincronizan gradientes y parámetros para mantener la convergencia.

6.6.2. CUDA: qué es y cómo se usa en el estado del arte

CUDA es la plataforma y el modelo de programación de NVIDIA para ejecutar cómputo paralelo en GPU. Proporciona un lenguaje de extensión de C/C++ y un conjunto de bibliotecas que exponen el hardware de forma estructurada. Un programa define *kernels* que se ejecutan de manera masiva en hilos agrupados en bloques; el planificador de la GPU distribuye esos bloques sobre *multiprocesadores* para aprovechar al máximo el paralelismo. El programador controla la organización de hilos, el acceso a memorias de distinta latencia y la sincronización cuando es necesario [51].

En el estado del arte de aprendizaje profundo, la mayoría de marcos de alto nivel,

como PyTorch y TensorFlow, encapsulan CUDA para que el usuario trabaje con tensores y operaciones declarativas. Bajo esa interfaz, las llamadas se resuelven en kernels altamente optimizados de cuDNN para convoluciones, normalizaciones y activaciones, y de cuBLAS o CUTLASS para operaciones de álgebra lineal. Además, estos marcos integran gestión de memoria, ejecución asíncrona y mezclas de precisión de forma automática, lo que simplifica el desarrollo de modelos y deja a CUDA el rol de motor de ejecución en segundo plano [49], [52], [53].

CAPÍTULO 7

Descripción de la colección de datos utilizada

Siguiendo lo determinado en la fase anterior, se optó por trabajar con una base de datos pública que reúne registros clínicos de EEG ya adquiridos, en lugar de realizar nuevas mediciones experimentales en el contexto de este proyecto. La detección de convulsiones con redes profundas requiere gran escala y diversidad de casos para que los modelos aprendan patrones robustos y generalicen entre pacientes y montajes. Bajo ese criterio, se seleccionó como eje principal el *Temple University Hospital EEG Corpus* (TUH/TUEG) y, en particular, su subconjunto de convulsiones *TUH EEG Seizure Corpus* (TUSZ), que proporciona etiquetas temporales precisas de eventos ictales y metadatos clínicos suficientes para construir ventanas y evaluar sensibilidad por episodio. [54]

7.1. Descripción breve de TUH y del subconjunto TUSZ

El corpus TUH/TUEG es una colección clínica a gran escala de EEG que reúne decenas de miles de estudios de práctica real, con documentación pública sobre formatos, nomenclatura de canales y guías de anotación [54], [55]. Para los eventos de convulsión, este trabajo empleó el subconjunto *TUH EEG Seizure Corpus* (TUSZ), que contiene anotaciones manuales de inicio y fin de convulsiones, información de canal y tipología de crisis, publicadas en versiones sucesivas que facilitan la reproducibilidad entre estudios [54], [56]. El uso de TUSZ permitió alinear de forma directa las etiquetas ictales con ventanas temporales y comparar arquitecturas y pérdidas centradas en sensibilidad ictal.

Como ejemplo de la información disponible, el Cuadro 2 resume algunas de las etiquetas de anotación incluidas en TUSZ y sus descripciones textuales.

Cuadro 2. Ejemplo de etiquetas de anotación disponibles en TUSZ

Índice	Etiqueta	Descripción
0	null	Anotación indefinida. No debería aparecer en los datos.
1	spsw	Evento epiléptico breve con un pico y/o onda lenta, de duración usualmente menor a 1 s.
2	gped	Descarga epiléptica periódica generalizada en múltiples regiones o hemisferios.
3	pled	Descarga epiléptica periódica lateralizada en una zona específica del cuero cabelludo.
4	eybl	Parpadeo. Artefacto de movimiento ocular con alta amplitud.
5	artf	Artefacto. Señal eléctrica no cerebral proveniente de equipo o entorno.
6	bckg	Otras señales cerebrales no asociadas a crisis epilépticas.
7	seiz	Crisis epiléptica. Anotación básica de convulsión.
8	fnsz	Crisis focal inespecífica localizada en una región cerebral.
9	gnsz	Crisis generalizada en gran parte o en todo el cerebro.
10	spsz	Crisis parcial simple. El paciente mantiene consciencia y capacidad de interacción.
11	cpsz	Crisis parcial compleja con alteración de la consciencia.
12	absz	Crisis de ausencia. Breve pérdida de atención, típica en niños.
13	tnsz	Crisis tónica. Rigidez muscular, a veces sin fase clónica.
14	cnsz	Crisis clónica. Sacudidas rítmicas sostenidas, usualmente en crisis tónico-clónicas.
15	tcsz	Crisis tónico-clónica con pérdida de consciencia y contracciones violentas.
16	atsz	Crisis atónica. Pérdida súbita del tono muscular.
17	mysz	Crisis mioclónica con espasmos o sacudidas breves.
18	nesz	Crisis no epiléptica sin signos electrográficos.
19	intr	Patrones inusuales o interesantes no clasificados.
20	slow	Disminución breve de la frecuencia.
21	eyem	Movimiento ocular, artefacto común en regiones frontales.
22	chew	Artefacto por masticación, de tipo “bursty” en múltiples canales.
23	shiv	Artefacto sostenido asociado a escalofríos o temblores.
24	musc	Artefacto muscular, de alta frecuencia asociada a nerviosismo.
25	elpp	“Electrode pop”. Picos simétricos por fallo en electrodos.
26	elst	Artefacto electrostático por movimiento o interferencia en electrodos.
27	calb	Artefacto por calibración de electrodos al inicio de los registros.
28	hphs	Periodo breve de ondas lentas de alta amplitud.
29	trip	Ondas trifásicas asociadas a condiciones metabólicas subyacentes.

Nota. El cuadro muestra un ejemplo de etiquetas y descripciones utilizadas en el *TUH EEG Seizure Corpus* (TUSZ) para anotar eventos y artefactos en EEG. Elaboración propia a partir [56].

7.2. Lineamientos de uso y consideraciones prácticas

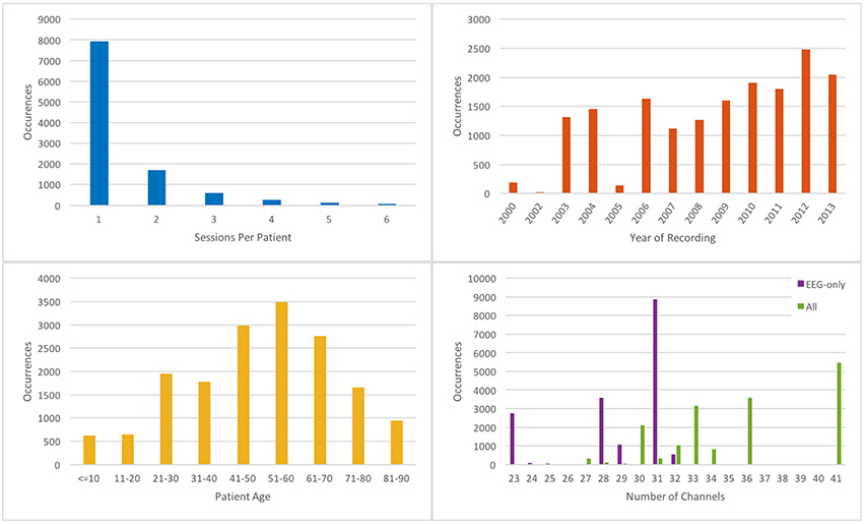
La distribución de TUH/TUSZ establece lineamientos de acceso y atribución, junto con guías técnicas para leer registros en formato EDF/EDF+ y recuperar correctamente anotaciones, listas de canales y montajes. Estas guías fueron la base para estandarizar el *pipeline* de lectura, construir conjuntos de entrenamiento/validación/prueba y reportar resultados comparables por paciente y por sesión [55], [56]. La disponibilidad de versiones documentadas y *changelogs* en la base de datos permite trazar la procedencia de cada archivo y sostener la reproducibilidad de experimentos.

7.3. Justificación frente a CHB–MIT y otras bases

La base *CHB–MIT Scalp EEG Database* es un referente clásico con población pediátrica, compuesta por 22 sujetos con episodios ictales registrados en condiciones hospitalarias controladas [57]. En contraste, TUSZ ofrece una escala mucho mayor y una variabilidad demográfica significativa, lo cual se refleja en la diversidad de sesiones por paciente, años de registro, edades y configuraciones de canales ilustradas en la Figura 5. Esta amplitud de condiciones reales, junto con la riqueza de anotaciones temporales, permite una evaluación inter-paciente más robusta y comparaciones más finas entre distintos tipos de crisis. [55]

Otras colecciones públicas pueden ser útiles como complemento, pero la combinación de volumen, detalle de etiquetas y documentación técnica de TUSZ la hace especialmente adecuada para entrenar y validar modelos que dependen de la alineación exacta evento a ventana.

Figura 5. Distribución del corpus TUH-EEG



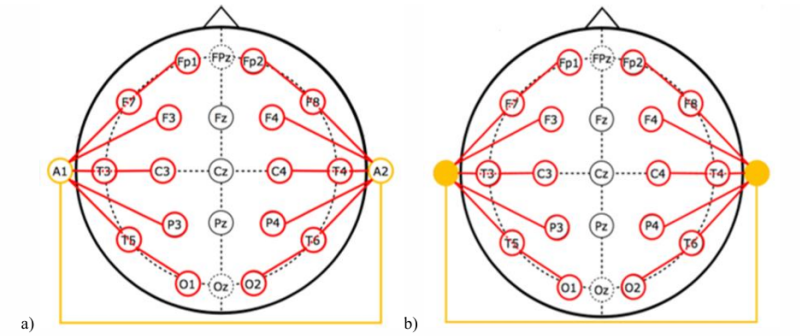
Nota: La figura muestra diferentes métricas del corpus TUH-EEG: histogramas de sesiones por paciente y por año, así como la distribución de edades y el número de canales registrados. Figura obtenida de [58].

CAPÍTULO 8

Preparación de los datos

Se trabajó con montajes bipolares definidos en TUH, en particular TCP_AR y TCP_-LE, que especifican pares bipolares y nomenclatura de canales sobre el sistema 10–20. Estas definiciones permitieron estandarizar el orden de canales y garantizar consistencia anatómica en los tensores de entrada. Para el conjunto final se priorizó el montaje AR por su mayor cobertura de pares locales y su utilidad práctica en la detección de transitorios ictales [55]. Las diferencias en la disposición de electrodos y pares bipolares para los montajes AR y LE se ilustran en la Figura 6.

Figura 6. Ubicación de los electrodos para dos montajes presentes en TUSZ: a) AR (Average Reference), b) LE (Linked Ears)



Nota: La figura ilustra la disposición estándar de electrodos según el sistema 10-20 utilizada en el corpus TUSZ, mostrando las diferencias entre el montaje de referencia promedio (AR) y el montaje con orejas enlazadas (LE). Imagen obtenida de [55].

8.1. Montajes empleados y criterio de selección

El método AR genera más pares bipolares, lo que significa que se obtienen más combinaciones de señales y, por tanto, más información en cada ventana de análisis. Sin embargo, su problema es que puede cambiar mucho de un registro a otro si faltan canales o se usan montajes distintos. El método LE, por el contrario, utiliza menos combinaciones, lo que lo hace más sencillo y estable en situaciones donde los datos se recolectan de formas diferentes, aunque a cambio se pierde detalle sobre la actividad en regiones específicas del cerebro. El enfoque AR-A busca un equilibrio, combinando referencias distintas aunque eso implique perder algunos pares. En la práctica, estas diferencias se resolvieron al imponer un orden común de los canales y revisar que todos estuvieran disponibles antes de procesar [55].

8.1.1. Formatos de señal y etiqueta

Los registros se leyeron desde EDF, formatos estándar que permiten codificar múltiples canales y anotar eventos, su especificación y uso en EEG clínico están ampliamente documentados [59], [60]. Las etiquetas de convulsión provinieron de archivos de anotación con marcas de inicio y fin de clasificación binaria para la presencia de eventos de convulsión, lo que posibilitó alinear ventanas con episodios ictales de forma reproducible y trazable [54], [55].

8.1.2. Acceso, organización y lineamientos

Se respetó la organización por `split` del corpus (`train/dev/eval`) y las guías públicas de TUH sobre nomenclatura, montajes y estructuras de directorio [54], [55]. El preprocesamiento verificó: i) presencia de los pares requeridos por el montaje, ii) tasas de muestreo esperadas y ausencia de canales *non-EEG* no deseados, iii) coherencia temporal entre señal y anotaciones. Cuando faltaron canales críticos o se detectaron inconsistencias temporales, el registro se excluyó para mantener la forma del tensor y la validez de las etiquetas [55], [60].

8.2. Construcción del conjunto de datos: ventaneo, remuestreo y normalización

8.2.1. Preprocesamiento

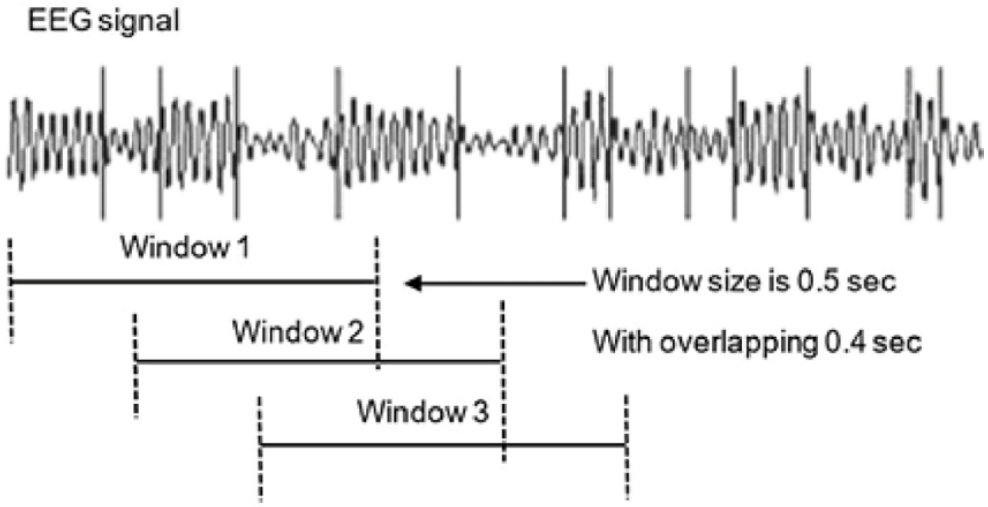
Se aplicó filtro pasa banda en el rango clínicamente relevante, filtro de muesca a la frecuencia de red, remuestreo para homogeneizar a una frecuencia objetivo y normalización por canal. En montajes bipolares, cada derivación se obtuvo por diferencia anódica–catódica antes del ventaneo. Estas prácticas son estándar en análisis de EEG y reducen variabilidad no informativa previa al aprendizaje. [28], [60]

8.2.2. Ventaneo deslizante

Se generaron ventanas temporales de longitud y salto configurables. Cada ventana se etiquetó como positiva si solapaba con un intervalo ictal, o con una secuencia de etiquetas por paso cuando se requirió granularidad fina. Este esquema es habitual en detección de eventos en series continuas y permite ajustar la sensibilidad mediante el grado de solapamiento, tal como se ilustra en la Figura 7. [28], [54]

En este contexto, se denomina segmento de interés a aquellas ventanas etiquetadas como ictales o que contienen actividad representativa de una crisis. La Figura 8 muestra un ejemplo de este tipo de ventana extraída de un registro EEG anotado, que ilustra el patrón de actividad convulsiva que se busca identificar de manera automática.

Figura 7. Ejemplo del funcionamiento de las ventanas deslizantes (*sliding windows*) en una señal de EEG

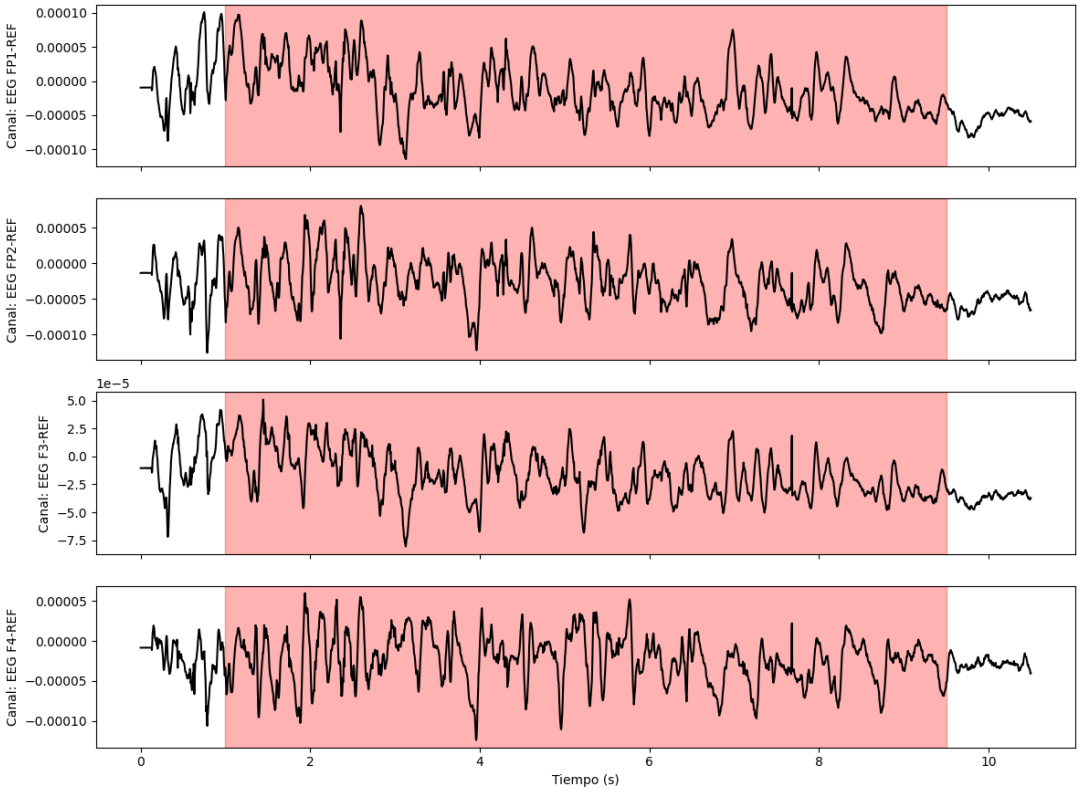


Nota: La imagen ilustra el proceso de segmentación temporal de una señal de EEG mediante *ventanas deslizantes*. Cada ventana tiene una duración de 0.5 segundos y se desplaza con un traslape de 0.4 segundos (80 %), lo cual permite conservar continuidad temporal y minimizar la pérdida de información en los límites de cada segmento. Imagen obtenida y adaptada de [61].

8.2.3. Balanceo y muestreo

El conjunto está desbalanceado porque las ventanas sin convulsión superan ampliamente a las ictales. Para evitar que el modelo aprenda a "acertar" prediciendo siempre la clase mayoritaria, se usaron dos niveles de corrección. Primero, al formar cada lote se controló la fracción de ejemplos positivos, de modo que el modelo viera casos ictales de manera regular durante todo el entrenamiento. Segundo, en la función de pérdida se aplicó ponderación por clase para penalizar más los errores sobre la clase minoritaria. En escenarios con desbalance marcado, también se consideraron pérdidas adaptadas (por ejemplo, focal o Tversky) que reducen la influencia de ejemplos triviales y concentran el aprendizaje en casos difíciles, lo cual mantiene la sensibilidad a episodios breves y poco frecuentes [34], [35]. En todos los

Figura 8. Ejemplo de segmento de interés en un registro EEG



Nota: El segmento mostrado corresponde a una ventana de interés extraída de un registro EEG anotado, en la cual se observa un ejemplo representativo de actividad convulsiva. Imagen de elaboración propia.

casos, el balanceo se implementó sin mezclar pacientes entre particiones, para no introducir sesgos en la validación.

8.2.4. Elección entre AR, LE y AR-A

Se compararon los montajes AR, LE y AR-A atendiendo a tres criterios: cobertura espacial efectiva, consistencia entre registros y compatibilidad con las etiquetas de convulsión. AR ofrece más pares bipolares locales y aporta señales con mayor detalle por ventana. LE reduce dimensionalidad y puede ser más estable cuando la colocación varía entre estudio. AR-A armoniza referencias alternativas a costa de perder algunos pares. Para el conjunto final se eligió AR porque maximiza la información útil por ventana y facilita estandarizar el tensor de entrada, mientras que LE y AR-A se mantuvieron como contraste metodológico para verificar robustez del pipeline según las guías publicadas por TUH [54], [55]. Cuando faltaron canales críticos en un registro, se excluyó o se marcó para evitar inconsistencias en la forma de los tensores.

8.3. Optimización del *pipeline*

8.3.1. TensorFlow: TFRecord y `tf.data`

Se serializaron las ventanas y metadatos a TFRecord y se consumieron con `tf.data` para sostener un flujo continuo hacia la GPU. La canalización aplicó lectura intercalada de archivos, parseo en paralelo, `cache` cuando el tamaño lo permitió, `batch` para agrupar ejemplos y `prefetch` para solapar la preparación del siguiente lote con el entrenamiento en curso. Este diseño reduce cuellos de botella y hace predecible el rendimiento entre épocas [62], [63]. Para acelerar el cómputo se activó entrenamiento en precisión mixta, que disminuye uso de memoria y tiempo en operaciones aritméticas, y, cuando fue pertinente, se compiló a grafo con `tf.function`/XLA para fusionar operaciones y reducir la sobrecarga de ejecución tras un costo inicial de compilación [50], [64]. En validaciones se controló que estas optimizaciones no alteraran la exactitud numérica relevante para las métricas clínicamente importantes.

8.3.2. PyTorch: caché y `DataLoader`

En PyTorch se implementó un `Dataset` que aplica el mismo preprocesado y ventaneo y que, en corridas repetidas, puede usar caché por partición para evitar trabajo redundante. El `DataLoader` empleó múltiples procesos para preparar lotes en paralelo, `pin_memory` para acelerar el envío de tensores a la GPU y prelectura para mantener alta ocupación del dispositivo [65]. Además, se activó precisión mixta para aprovechar primitivas de GPU, y, cuando fue útil, se compiló el modelo con `torch.compile` para generar núcleos optimizados y reducir la sobrecarga del intérprete durante el entrenamiento [66]. Estas mejoras se apoyan en bibliotecas subyacentes como cuDNN y cuBLAS, que proporcionan implementaciones eficientes de convoluciones y álgebra lineal utilizadas por ambas plataformas [49].

8.3.3. Otras optimizaciones

Se habilitó precisión mixta para reducir consumo de memoria y acelerar la aritmética en GPU [50]. Tanto TensorFlow como PyTorch aprovecharon cuDNN/cuBLAS para primitivas de convolución y álgebra lineal [49]. La compilación a grafo (XLA/`tf.function` y `torch.compile`) fusionó operaciones y disminuyó sobrecarga de ejecución tras el costo inicial de compilación [64], [66]. Además, se añadieron controles de reproducibilidad (semillas, registro de versiones), validación estratificada por paciente en los distintos conjuntos de datos (entrenamiento, validación y evaluación) y un catálogo de metadatos por archivo (frecuencia de muestreo, número de canales, montaje, duración) para auditoría.

8.4. Características para conjuntos de datos de EEG y su integración en el aprendizaje profundo

8.4.1. Uso de características en redes profundas

Aunque las redes profundas pueden aprender representaciones directamente de la señal cruda, en EEG es habitual complementar el entrenamiento con características calculadas sobre ventanas deslizantes. Estas variables resumen la forma de onda, el contenido espectral, las relaciones entre canales y la dinámica temporal. Su utilidad práctica es triple: facilitan la convergencia al presentar señales con mejor relación señal a ruido, favorecen la interpretación por su vínculo con fenómenos neurofisiológicos y permiten combinar la serie cruda con descriptores adicionales mediante fusión temprana o fusión tardía [27], [28]. En este trabajo las características se calcularon por ventana y por canal o bien agregadas entre canales. Luego se normalizaron de manera consistente e ingresaron al modelo como tensores concatenados a la entrada en la fusión temprana o como vectores procesados en una rama paralela que se une a la representación aprendida antes de la clasificación en la fusión tardía. La elección depende de la arquitectura y del equilibrio entre costo computacional y ganancia en desempeño.

8.4.2. Características en el dominio del tiempo

En cada ventana se calcularon medidas como media, varianza, desviación estándar, asimetría, curtosis, energía, cruces por cero, longitud de línea y los parámetros de Hjorth que incluyen actividad, movilidad y complejidad. Estos descriptores capturan amplitud, irregularidad y cambios rápidos que suelen aparecer al inicio de un episodio ictal. La literatura reporta su utilidad de forma aislada y también en combinación con rasgos espectrales [27], [28], [67]. Para su integración cada estadístico se obtiene por canal dentro de la ventana y se reúne en un vector multicanal que se normaliza con un z score robusto. En fusión temprana el vector se proyecta a un pequeño mapa que se concatena como canales auxiliares junto a la señal. En fusión tardía el vector pasa por capas densas y se concatena con la representación de la red antes de la etapa de decisión.

8.4.3. Características en el dominio de la frecuencia

Se utilizó la densidad espectral de potencia para extraer potencia absoluta y relativa en bandas delta, theta, alfa, beta y gamma. Además se calcularon centroide espectral, frecuencia dominante y entropía espectral. Estas variables resumen la distribución de ritmos y suelen ser menos sensibles a picos transitorios, por lo que resultan útiles en la clasificación entre actividad ictal y no ictal [27], [28]. La densidad espectral se estimó con el método de Welch o con la transformada de corto tiempo. Posteriormente se integró la potencia en cada banda y se normalizó por la energía total de la ventana. El resultado es un vector por canal que puede concatenarse a la entrada. Cuando se dispone de mapas tiempo frecuencia es posible promediar en subbandas y utilizar esa representación reducida en una rama convolucional bidimensional.

8.4.4. Características en el tiempo frecuencia

Las representaciones tiempo frecuencia, como la transformada de corto tiempo o las *wavelets* de Morlet, permiten observar cómo cambia la energía por bandas a lo largo del tiempo. Estas herramientas resultan adecuadas para detectar estallidos y patrones no estacionarios que pueden anticipar el inicio de una convulsión. A partir de estos mapas se pueden extraer momentos por banda y por tiempo o bien descriptores de textura [27], [28]. Para su integración se calcula un mapa tiempo frecuencia por canal, se reduce a un conjunto de subbandas y se normaliza. El tensor resultante con dimensiones de bandas por tiempo por canales puede ingresar a una rama convolucional bidimensional en paralelo a la rama temporal unidimensional que procesa la serie cruda.

8.4.5. Características de conectividad y relaciones entre canales

Medidas como la coherencia, el *phase lag index* y la correlación cruzada describen la sincronización entre regiones. Este tipo de información resulta valiosa cuando la actividad ictal se propaga por una red de canales [28]. En la práctica, estas medidas se estiman por pares de canales y luego se agregan por regiones del sistema 10–20, entendidas como agrupaciones de electrodos que corresponden aproximadamente a lóbulos o áreas anatómicas (por ejemplo, frontal, temporal, parietal u occipital), para contener la dimensionalidad. Puede utilizarse un resumen por percentiles o una proyección por componentes principales. El resultado se incorpora como vector de conectividad en fusión tardía o como matrices que alimentan una rama convolucional bidimensional.

8.4.6. Características de la dinámica y espacio de Koopman

En este trabajo, la familia de métodos inspirados en la teoría de Koopman se utiliza para extraer características dinámicas a partir de ventanas multicanal de EEG. Para cada ventana se organizan fotogramas consecutivos de la señal y se aplica DMD para estimar la mejor evolución lineal y su espectro asociado.

De ese análisis se extraen características como las partes real e imaginaria de los autovalores, que informan sobre frecuencia y crecimiento o decaimiento, así como las amplitudes de los modos y la energía del modo dominante. Estas variables se normalizan e ingresan como vector en fusión tardía o se emplean como reducción de dimensión previa a un clasificador. Cuando se requiere mayor capacidad, la variante EDMD permite ampliar el conjunto de observables con funciones no lineales y capturar estructuras dinámicas más ricas [30]. Con esto:

8.4.7. Incorporación en el conjunto de datos y en el entrenamiento

El esquema del conjunto de datos almacena para cada ventana el tensor preprocesado del montaje AR ya normalizado y un vector de características que puede incluir rasgos de tiempo, frecuencia, tiempo frecuencia, conectividad y Koopman. También se guardan metadatos como identificador de paciente, frecuencia de muestreo y descripción del preprocesamiento,

además de la etiqueta en forma dura o secuencial. En TensorFlow, este paquete se serializa en TFRecord [62] y se procesa con la API tf.data mediante lectura intercalada, parseo en paralelo, caché y *prefetch* [63]. En PyTorch se utiliza un sistema de caché por partición y se alimenta un *DataLoader* con múltiples procesos, memoria anclada y prelectura [65]. Respecto a la integración con el modelo, en la fusión temprana los vectores o mapas se concatenan a la entrada y se procesan junto a la serie. En la fusión tardía la señal cruda y las características siguen ramas separadas, por ejemplo una temporal y otra densa, y se unen antes de la capa de clasificación. En ambos casos se mantuvieron normalizaciones coherentes entre particiones y, cuando hubo desbalance de clases, se combinó con ponderación por clase o con pérdidas adaptadas [27]. Para acelerar el entrenamiento se habilitó precisión mixta [50] y, cuando fue pertinente, la compilación a grafo con XLA en TensorFlow [64] o con la función torch.compile en PyTorch [66], apoyándose en primitivas eficientes de cuDNN [49].

CAPÍTULO 9

Redes propuestas para detección de segmentos de interés en EEG y metodología de entrenamiento

En este capítulo se describen las arquitecturas de redes profundas propuestas para la detección de segmentos de interés en EEG y la metodología de entrenamiento empleada para su ajuste y evaluación. A partir del conjunto de datos y del *pipeline* de preprocesamiento definidos en capítulos anteriores, se detallan tres familias de modelos: una *Temporal Convolutional Network* (TCN) especializada en capturar dependencias temporales largas, una red híbrida que combina convoluciones temporales con unidades recurrentes bidireccionales y atención, y un *Transformer* temporal basado en auto-atención. Además, se presentan los elementos comunes de estas arquitecturas (normalización, regularización y manejo de desbalance de clases) y se organiza el flujo de entrenamiento en etapas claramente reproducibles, desde la construcción del conjunto de ventanas hasta la selección de umbrales a partir de curvas ROC y Precisión–*Recall*, preparando el marco necesario para la interpretación de los resultados del capítulo siguiente.

9.1. TCN optimizada para EEG

La *Temporal Convolutional Network* (TCN) emplea convoluciones causales y dilatadas para cubrir intervalos largos sin perder resolución temporal. Esto favorece la detección de irrupciones ictales al mantener la alineación exacta entre señales de entrada y salidas. La arquitectura propuesta incluye: (i) convoluciones separables en profundidad para reducir cómputo, (ii) normalización y activaciones suaves (por ejemplo, GELU) para estabilizar el entrenamiento, (iii) bloques de realce por canal (*squeeze-and-excitation*) que priorizan derivaciones informativas y (iv) atajos residuales para mejorar el flujo de gradiente. Un bloque multiescala agrega *kernels* de distinta longitud para capturar tanto descargas rápidas como modulaciones más lentas, lo que resulta útil en clasificación binaria convulsión/no

convulsión.

9.2. Red híbrida CNN temporal + RNN (BiLSTM/GRU) con atención

Este modelo combina un frente convolucional 1D separable (que extrae rasgos morfológicos por canal con bajo costo) con una rama recurrente bidireccional (LSTM/GRU) que modela dependencias de largo alcance propias de la progresión ictal. Se añade un mecanismo de atención que permite enfocar los intervalos más informativos antes de la decisión. La cabeza de clasificación puede operar por ventana o por paso temporal, permitiendo ajustar el compromiso entre precisión y cobertura alrededor del inicio de las crisis.

9.3. *Transformer* temporal

Los *Transformers* temporales modelan la serie como una secuencia de *patches* o marcos y aprenden relaciones entre todos los instantes mediante atención multi-cabeza. Esto permite resaltar segmentos informativos y su contexto a distancia, algo útil cuando la actividad ictal se propaga o cambia de frecuencia. En ventanas de EEG multicanal, una proyección lineal inicial empaqueta el tiempo y los canales en *tokens*. Tras varias capas de auto-atención y MLP, una cabeza de clasificación produce la probabilidad ictal.

9.4. Elementos comunes de las redes

Ambas arquitecturas comparten componentes estándar de redes profundas: normalización (por lote o por capa), *dropout* para regularización, activaciones no lineales (ReLU/GELU), atajos residuales y una cabeza de clasificación adecuada para datos desbalanceados (ponderación por clase o muestreo balanceado). Estas prácticas, ampliamente recomendadas en la literatura moderna de aprendizaje profundo, mejoran estabilidad numérica, generalización y velocidad de convergencia en EEG.

9.5. Metodología de entrenamiento

El flujo de trabajo se organiza en cinco etapas principales: preparación previa, construcción del conjunto de datos, construcción del modelo, entrenamiento con registro sistemático de métricas y, finalmente, evaluación y obtención de resultados. Todas estas etapas se controlan desde archivos de configuración y dejan un registro en una carpeta de salida, de modo que cada ejecución pueda repetirse y auditarse. La Figura 9 resume de forma esquemática estas etapas y sus relaciones, desde la configuración inicial hasta la evaluación final del modelo.

En la fase de preparación previa se asume que la base de datos de EEG ya fue descargada y organizada en particiones de entrenamiento, validación y evaluación. Además, se dispone de un archivo de configuración donde se indican las rutas de cada partición, el tipo de modelo que se va a usar, los parámetros de preprocesamiento (montaje, filtros, frecuencia de muestreo objetivo, tamaño y salto de ventana, entre otros) y los parámetros de entrenamiento (tamaño de lote, número de épocas, función de pérdida, optimizador y métricas de interés). Antes de iniciar una corrida, el sistema lee este archivo, comprueba que las rutas existan y crea una carpeta específica para almacenar la configuración utilizada, información del entorno y todos los resultados que se generen.

A partir de esa configuración comienza la construcción del conjunto de datos. Primero se identifican los registros EDF que pertenecen a cada partición y se asocian a pacientes concretos, para evitar mezclar datos de la misma persona entre entrenamiento y evaluación. Luego se leen los registros seleccionados y se proyectan al montaje bipolar elegido, por ejemplo AR o LE, obteniendo derivaciones anódica–catódica sobre el sistema 10–20. En este punto se revisa que los canales requeridos estén presentes y que la frecuencia de muestreo sea la esperada; si un registro no cumple estas condiciones, se descarta para no introducir inconsistencias.

Una vez proyectadas las señales al montaje deseado se aplica el preprocesamiento. Cada registro multicanal pasa por un filtro pasa banda que conserva el rango de interés clínico, por un filtro de muesca para suprimir la frecuencia de la red eléctrica y por un remuestreo que homogeneiza todas las señales a una misma frecuencia objetivo. Después se normaliza cada canal, por ejemplo ajustando su media y desviación estándar, de modo que las amplitudes queden en escalas comparables. Este conjunto de transformaciones deja las señales en un formato uniforme y con menos ruido antes de segmentarlas.

El siguiente paso es el ventaneo deslizante. Cada registro preprocesado se divide en ventanas de duración fija, con un traslape definido entre una y otra. Para cada ventana se consultan las anotaciones de crisis asociadas al registro: si la ventana se solapa con un intervalo ictal se etiqueta como positiva, y si solo contiene actividad de fondo se etiqueta como negativa. Cuando se necesita más detalle, el mismo mecanismo permite guardar etiquetas internas por paso de tiempo dentro de la ventana. Además, el flujo puede controlar la proporción de ventanas positivas que entran en entrenamiento para evitar que la clase "sin crisis" domine por completo el conjunto de ejemplos. Si la configuración lo indica, sobre cada ventana también se calculan características adicionales, como descriptores en el dominio del tiempo y la frecuencia, medidas de conectividad entre canales o variables derivadas de descomposiciones dinámicas. Estas características se guardan junto a la ventana cruda y más adelante pueden incorporarse a una rama secundaria del modelo.

Al terminar esta etapa, las ventanas, sus etiquetas, las características adicionales y la información de paciente y registro se almacenan en estructuras reutilizables para cada partición. La idea es que el trabajo pesado de leer, filtrar, remuestrear y ventanear se haga una vez, y que los entrenamientos posteriores solo tengan que reutilizar estos paquetes sin repetir el preprocesamiento desde cero.

Con los datos ya preparados, el flujo construye el modelo. A partir de la configuración se decide si se utilizará una red TCN, una red híbrida o un *Transformer*. La forma de la entrada principal se fija según el número de canales y la longitud de cada ventana; si se

han habilitado características adicionales, se define también una entrada para esos vectores. La salida del modelo se diseña para producir una probabilidad de crisis por ventana (o por paso de tiempo, si se trabaja a mayor resolución). La función de pérdida se elige de acuerdo con las necesidades del problema: puede ser entropía cruzada estándar, una variante focal que enfatiza ejemplos difíciles o una pérdida basada en índices de similitud como Tversky, incluyendo ponderaciones para atender el desbalance entre clases. En esta misma etapa se selecciona el algoritmo de optimización, típicamente una variante de Adam o AdamW con una tasa de aprendizaje inicial, un esquema de decaimiento de pesos y otros hiperparámetros definidos en el archivo de configuración. Cuando el hardware lo permite, el flujo activa el uso de precisión mixta y mecanismos de compilación a grafo para aprovechar mejor la GPU.

El entrenamiento comienza construyendo flujos de datos separados para entrenamiento y validación a partir del conjunto de ventanas ya procesado. Para entrenamiento se generan lotes que se barajan en cada época y que, si así se configuró, incluyen una fracción controlada de ventanas positivas para que la red vea con regularidad ejemplos ictales. Para validación se utilizan los mismos datos, pero sin barajado, para obtener medidas de referencia estables. Cada época consiste en recorrer los lotes de entrenamiento, calcular las predicciones, actualizar los parámetros para reducir la pérdida y, al final, evaluar el modelo sobre la partición de validación. Durante este proceso se registran de forma continua la pérdida de entrenamiento y de validación, así como métricas de clasificación en validación, como exactitud, sensibilidad, especificidad, precisión, F_1 y áreas bajo las curvas ROC y Precisión–Recuperación. También se miden tiempos de cómputo por lote y por época, y un rendimiento aproximado en muestras por segundo. Todo este historial se guarda en archivos tabulares dentro de la carpeta de la corrida para poder analizar después cómo evolucionó el modelo.

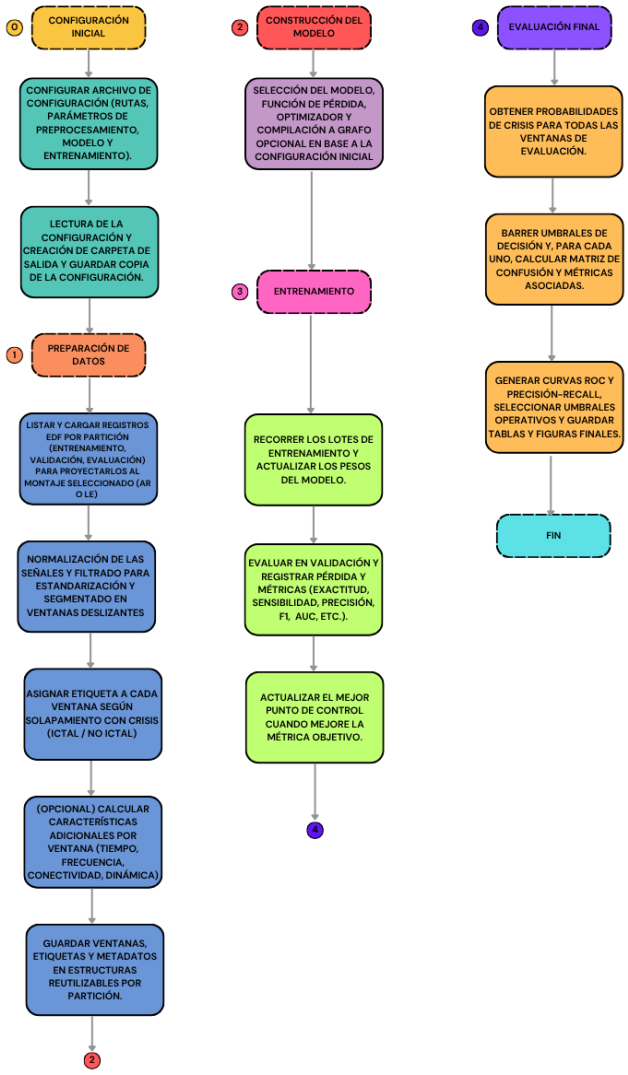
Mientras avanza el entrenamiento, el flujo vigila una métrica principal, normalmente alguna que refleje bien el comportamiento sobre la clase de crisis, como la sensibilidad o el área bajo la curva Precisión–Recuperación. Siempre que esta métrica mejora, se guarda un punto de control con los pesos actuales del modelo como "mejor estado hasta el momento". Si pasan varias épocas sin mejora, se puede aplicar una parada temprana para evitar sobreajuste y ahorrar tiempo de cómputo. Al terminar, en la carpeta de salida se conserva tanto el historial completo como el mejor conjunto de pesos encontrado.

La evaluación final se realiza sobre la partición de prueba o evaluación, reutilizando las mismas ventanas y etiquetas que se construyeron durante el preprocesamiento. Primero se reconstruye el modelo con la misma arquitectura y parámetros de entrada y se cargan los pesos del mejor punto de control. Luego el modelo procesa todas las ventanas de evaluación y produce para cada una una probabilidad de crisis. Estas probabilidades se guardan en archivos intermedios, lo que permite experimentar después con distintos umbrales de decisión sin tener que volver a ejecutar la inferencia.

A partir de estas probabilidades se explora un rango de umbrales posibles. Para cada umbral se determina qué ventanas se consideran positivas o negativas y se calcula la matriz de confusión, con sus verdaderos positivos, verdaderos negativos, falsos positivos y falsos negativos. A partir de allí se obtienen las métricas habituales: exactitud, sensibilidad, especificidad, precisión y F_1 para cada punto del rango. Con todos estos valores se trazan las curvas ROC y Precisión–Recuperación y se calculan sus áreas bajo la curva, que sirven como resumen global del comportamiento del modelo. Finalmente se escogen uno o varios umbrales de interés, por ejemplo aquellos que maximizan cierta combinación de sensibilidad

y precisión, y se generan las tablas y figuras que recogen las métricas finales y la matriz de confusión asociada. Estos resultados son los que se incorporan en el análisis comparativo de arquitecturas y en la discusión sobre la utilidad práctica de cada modelo.

Figura 9. Diagrama general de la metodología propuesta



Nota: El diagrama resume el flujo completo de la metodología empleada, desde la configuración inicial y la preparación de datos, hasta la construcción, entrenamiento y evaluación del modelo. Elaboración propia.

CAPÍTULO 10

Resultados de entrenamientos

En este capítulo se presentan los resultados de los experimentos de entrenamiento realizados en tres entornos de cómputo: una estación de trabajo de los laboratorios de la universidad, una *High Performing Computer* (HPC) de la UVG y una instancia en la nube *Amazon EC2*. En cada plataforma se evaluaron todas las arquitecturas, variando únicamente los recursos de hardware disponibles, cuyos detalles se resumen en el Cuadro 3. El objetivo fue evaluar el impacto de las optimizaciones propuestas sobre la velocidad de entrenamiento y sobre las métricas de evaluación de las redes, comparando el comportamiento entre los distintos entornos y analizando hasta qué punto las mejoras de infraestructura se traducen en ganancias efectivas de desempeño.

Cuadro 3. Recursos de cómputo utilizados en los entrenamientos

Entorno	GPU	Memoria RAM
Computadora de laboratorio	*NVIDIA T400 (4 GB)*	*32 GB*
HPC UVG	NVIDIA QUADRO P5000 (16 GB)	256 GB
Instancia Amazon EC2	NVIDIA T4 (16 GB)	64 GB

Nota. La tabla resume las características principales de los tres entornos de cómputo empleados para los experimentos de entrenamiento, indicando únicamente el tipo de GPU y la memoria RAM disponible en cada caso. Los valores mostrados son de referencia y deben completarse según la configuración efectiva utilizada. Elaboración propia.

10.1. Mejores resultados por arquitectura

En esta sección se presentan los mejores resultados obtenidos para cada arquitectura en los distintos entornos de cómputo evaluados. Las tres redes se entrenaron y ajustaron tanto

en la estación de laboratorio como en la HPC y en la instancia Amazon EC2, con el fin de comparar su desempeño bajo diferentes recursos de hardware. Los mejores modelos de la TCN y de la red híbrida se obtuvieron en la computadora de laboratorio, mientras que la configuración más competitiva del Transformer se alcanzó en la instancia EC2, donde fue posible explorar variantes con mayor demanda computacional.

10.1.1. Resultados de la TCN

A continuación se resume un experimento con la TCN descrita, usando ventanas de 5 s (salto 2.5 s), filtrado 0.5–40 Hz, muesca 60 Hz, remuestreo a 256 Hz, pérdida focal y un *stream* balanceado (20 %). El umbral de decisión ilustrativo fue 0.348 (punto marcado en las curvas).

Cuadro 4. Métricas globales de la TCN para el umbral ilustrado

	Exactitud	**Sensibilidad**	**Especificidad**	**Precisión**	**F1**	**AUC ROC / AP**
TCN (umbral 0.348)	0.791	0.581	0.806	0.178	0.273	0.707 / 0.264

Nota. La tabla resume las métricas globales obtenidas por la red TCN en el conjunto de evaluación para un umbral de decisión de 0.348. Se reportan exactitud, sensibilidad, especificidad, precisión, F1 y el área bajo la curva ROC (AUC ROC), junto con el área bajo la curva Precisión–Recuperación (AP). Elaboración propia.

Cuadro 5. Matriz de confusión de la TCN

		Predicción	
		Sin crisis	**Crisis**
Verdadero	**Sin crisis**	**80.59 %**	19.41 %
	Crisis	41.88 %	**58.12 %**

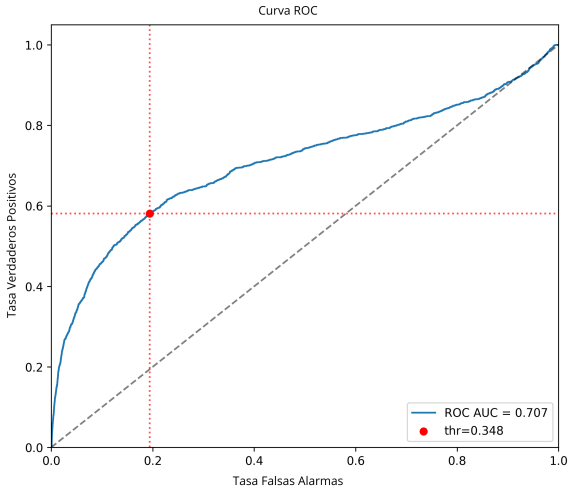
Nota. La tabla muestra la matriz de confusión de la red TCN en términos porcentuales, donde se desglosan las proporciones de ventanas correctamente clasificadas y mal clasificadas para las clases *Crisis* y *Sin crisis*. Elaboración propia.

La AUC ROC de 0.707 en la Figura 10 muestra una capacidad discriminativa moderada a lo largo de distintos umbrales, coherente con los valores globales reportados en el Cuadro 4. Con el umbral ilustrado (0.348), la sensibilidad alcanza 0.581 y la especificidad 0.806, con precisión de 0.178 y $F1 = 0.273$, mientras que la matriz de confusión de el Cuadro 5 evidencia que el modelo acierta la mayoría de las ventanas sin crisis, pero aún omite una fracción relevante de ventanas ictales. La curva Precisión–*Recall* de la Figura 11, con un área bajo la curva (AP) de 0.264, refleja la dificultad adicional que introduce el desbalance de clases.

En términos operativos, este ajuste tiende a ser más conservador, ya que prioriza una tasa de falsos positivos moderada a costa de perder parte de los eventos (sensibilidad menor que la de la arquitectura híbrida analizada en la sección previa). En escenarios clínicos donde la prioridad es no omitir convulsiones, esta configuración de TCN requeriría desplazar el umbral hacia una mayor sensibilidad utilizando las curvas de la Figura 10 y la Figura 11

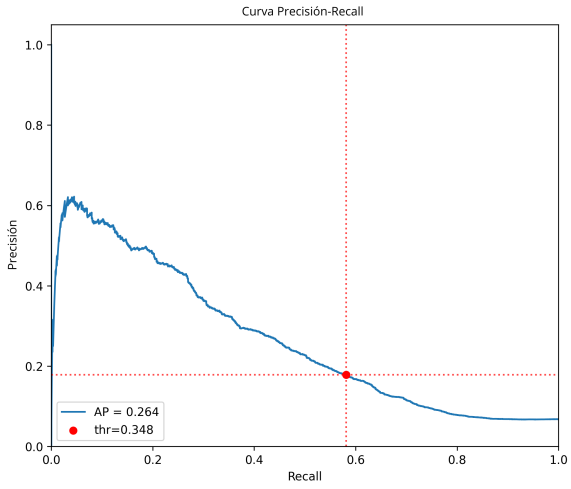
como guía, o complementar con posprocesamiento para consolidar ventanas positivas por episodio y reducir disparos aislados.

Figura 10. Curva ROC del modelo TCN



Nota. La figura muestra la curva ROC (*Receiver Operating Characteristic*) del modelo TCN en el conjunto de evaluación. El área bajo la curva (AUC) es 0.707 y el punto marcado en rojo corresponde al umbral operativo de 0.348, utilizado para calcular las métricas globales reportadas en el Cuadro 4. El eje horizontal corresponde a la tasa de falsos positivos (*False Positive Rate*) y el eje vertical a la tasa de verdaderos positivos (*True Positive Rate*). Elaboración propia.

Figura 11. Curva Precisión–*Recall* del modelo TCN



Nota. La figura muestra la curva Precisión–*Recall* del modelo TCN en el conjunto de evaluación. El área bajo la curva (AP) es 0.264 y se indica el mismo umbral operativo de 0.348 utilizado en la curva ROC, lo que permite comparar el comportamiento del modelo frente a distintos compromisos entre sensibilidad y precisión. El eje horizontal corresponde a la *Recall* (sensibilidad) y el eje vertical a la *Precision* (valor predictivo positivo). Elaboración propia.

10.1.2. Resultados de la red híbrida

A continuación se resume un experimento con la red híbrida descrita, usando ventanas de 5 s (salto 2.5 s), filtrado 0.5–40 Hz, muesca 60 Hz, remuestreo a 256 Hz, pérdida focal y un *stream* balanceado (20 %). El umbral de decisión ilustrativo fue 0.474 (punto marcado en las curvas).

Cuadro 6. Métricas globales de la red híbrida para el umbral ilustrado

	Exactitud	**Sensibilidad**	**Especificidad**	**Precisión**	**F1**	**AUC ROC / AP**
Híbrida (umbral 0.474)	0.770	0.616	0.781	0.169	0.265	0.735 / 0.225

Nota. La tabla resume las métricas globales obtenidas por la red híbrida en el conjunto de evaluación para un umbral de decisión de 0.474. Se reportan exactitud, sensibilidad, especificidad, precisión, F1 y el área bajo la curva ROC (AUC ROC), junto con el área bajo la curva Precisión–Recuperación (AP). Elaboración propia.

Cuadro 7. Matriz de confusión del modelo híbrido

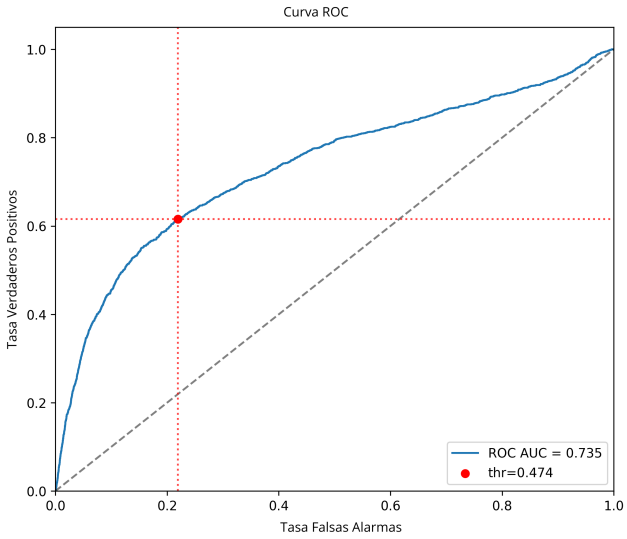
		Predicción	
		Sin crisis	**Crisis**
Verdadero	**Sin crisis**	**78.07 %**	21.93 %
	Crisis	38.41 %	**64.59 %**

Nota. La tabla presenta la matriz de confusión del modelo híbrido en términos porcentuales, mostrando la proporción de ventanas clasificadas correctamente y aquellas mal clasificadas entre las clases *Crisis* y *Sin crisis*. Elaboración propia.

La AUC ROC de 0.735 mostrada en la Figura 12 sugiere una capacidad discriminativa consistente en un rango amplio de umbrales, en concordancia con los valores globales de el Cuadro 6. Al fijar el umbral en 0.474, la sensibilidad alcanza 0.616 y la especificidad 0.781, mientras que la precisión se mantiene baja (0.169) debido a la prevalencia reducida de ventanas positivas y al coste asociado a los falsos positivos. La matriz de confusión de el Cuadro 7 muestra que el modelo acierta una fracción mayor de ventanas ictales que la TCN, a costa de incrementar ligeramente la proporción de falsas alarmas sobre la clase *Sin crisis*.

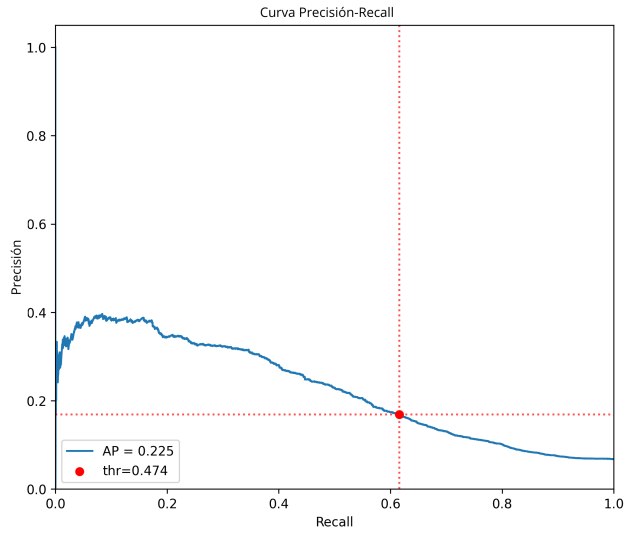
Por su parte, la curva Precisión–*Recall* de la Figura 13, con un AP de 0.225, refleja el compromiso entre sensibilidad y precisión en un entorno desbalanceado. En aplicaciones clínicas, suele ser necesario ajustar el umbral para elevar la precisión o combinar el modelo con etapas de posprocesamiento que consoliden ventanas cercanas en episodios y filtren disparos aislados, de modo que se reduzcan alarmas espurias manteniendo la sensibilidad.

Figura 12. Curva ROC del modelo híbrido



Nota. La figura muestra la curva ROC (*Receiver Operating Characteristic*) del modelo híbrido en el conjunto de evaluación. El área bajo la curva (AUC) es 0.735 y el punto marcado en rojo corresponde al umbral operativo de 0.474, utilizado para calcular las métricas globales reportadas en el Cuadro 6. El eje horizontal corresponde a la tasa de falsos positivos (*False Positive Rate*) y el eje vertical a la tasa de verdaderos positivos (*True Positive Rate*). Elaboración propia.

Figura 13. Curva Precisión–*Recall* del modelo híbrido



Nota. La figura muestra la curva Precisión–*Recall* del modelo híbrido en el conjunto de evaluación. El área bajo la curva (AP) es 0.225 y se indica el mismo umbral operativo de 0.474 utilizado en la curva ROC, lo que permite comparar el comportamiento del modelo frente a distintos compromisos entre sensibilidad y precisión. El eje horizontal corresponde a la *Recall* (sensibilidad) y el eje vertical a la *Precision* (valor predictivo positivo). Elaboración propia.

10.1.3. Resultados de la red *Transformer*

A continuación se resume un experimento con la red *Transformer* descrita, usando ventanas de 5 s (salto 2.5 s), filtrado 0.5–40 Hz, muesca 60 Hz, remuestreo a 256 Hz, pérdida focal y un *stream* balanceado (20 %). El umbral de decisión operativo se seleccionó a partir de las curvas ROC y Precisión–*Recall* y se corresponde con el punto marcado en rojo en las Figuras 14 y 15.

Cuadro 8. Métricas globales de la red *Transformer* para el umbral ilustrado

	Exactitud	**Sensibilidad**	**Especificidad**	**Precisión**	**F1**	**AUC ROC / AP**
Transformer	0.770	0.344	0.801	0.111	0.168	0.678 / 0.103

Nota. La tabla resume las métricas globales obtenidas por la red *Transformer* en el conjunto de evaluación para el umbral operativo seleccionado. Se reportan exactitud, sensibilidad, especificidad, precisión, F1 y el área bajo la curva ROC (AUC ROC), junto con el área bajo la curva Precisión–Recuperación (AP). Elaboración propia.

Cuadro 9. Matriz de confusión del modelo *Transformer*

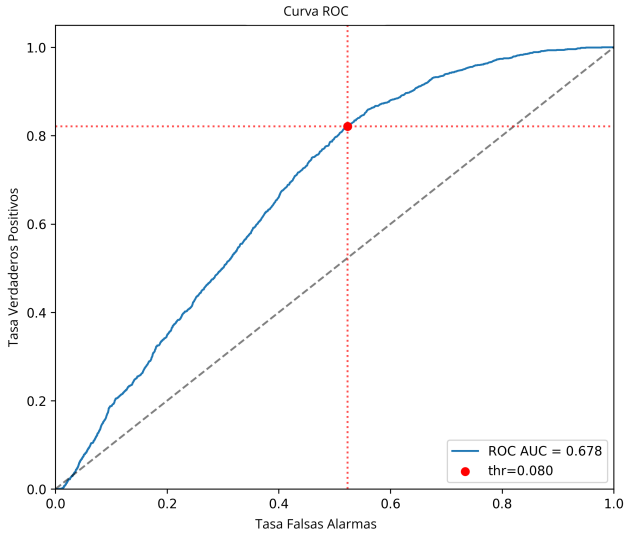
		Predicción	
		Sin crisis	**Crisis**
Verdadero	**Sin crisis**	**80.11 %**	19.89 %
	Crisis	**65.61 %**	34.39 %

Nota. La tabla presenta la matriz de confusión del modelo *Transformer* en términos porcentuales por fila. Cada fila suma 100 % y muestra, para las clases *Crisis* y *Sin crisis*, la proporción de ventanas clasificadas correctamente y aquellas mal clasificadas. Elaboración propia.

La AUC ROC de 0.678 observada en la Figura 14 indica una capacidad discriminativa moderada, coherente con las métricas agregadas de el Cuadro 8. Con el umbral operativo seleccionado, la sensibilidad alcanza 0.344 y la especificidad 0.801, mientras que la precisión se mantiene baja (0.111) debido a la prevalencia reducida de ventanas positivas y al número de falsos positivos. La matriz de confusión de el Cuadro 9 confirma que el modelo tiende a clasificar correctamente la mayoría de las ventanas sin crisis, pero omite una proporción importante de ventanas ictales.

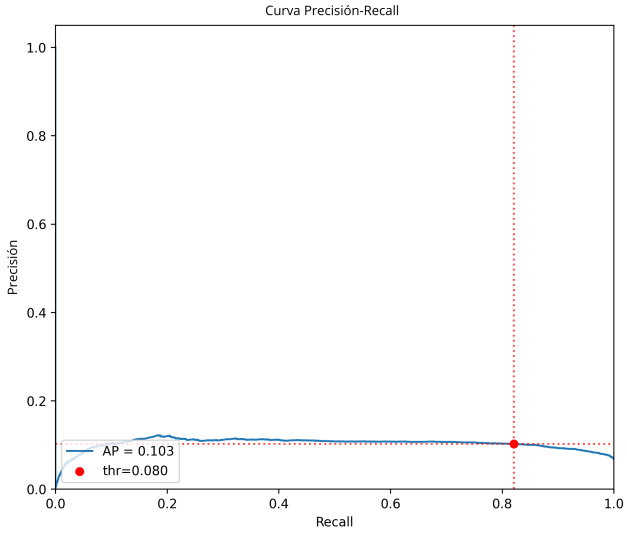
Por su parte, la curva Precisión–*Recall* de la Figura 15, con un AP de 0.103, refleja la dificultad del *Transformer* para mantener precisión alta en presencia de un desbalance pronunciado entre clases. En escenarios clínicos, este comportamiento implicaría la necesidad de ajustar el umbral hacia valores más sensibles y/o incorporar etapas de posprocesamiento que agrupen ventanas positivas en episodios y filtren activaciones aisladas, con el fin de mejorar la utilidad práctica del modelo.

Figura 14. Curva ROC del modelo *Transformer*



Nota. La figura muestra la curva ROC (*Receiver Operating Characteristic*) del modelo *Transformer* en el conjunto de evaluación. El área bajo la curva (AUC) es aproximadamente 0.678 y el punto marcado en rojo corresponde al umbral operativo utilizado para calcular las métricas globales de el Cuadro 8. El eje horizontal corresponde a la tasa de falsos positivos (*False Positive Rate*) y el eje vertical a la tasa de verdaderos positivos (*True Positive Rate*). Elaboración propia.

Figura 15. Curva Precisión–*Recall* del modelo *Transformer*



Nota. La figura muestra la curva Precisión–*Recall* del modelo *Transformer* en el conjunto de evaluación. El área bajo la curva (AP) es aproximadamente 0.103 y se indica el mismo umbral operativo que en la Figura 14, lo que permite analizar el compromiso entre sensibilidad y precisión en un entorno desbalanceado. El eje horizontal corresponde a la *Recall* (sensibilidad) y el eje vertical a la *Precision* (valor predictivo positivo). Elaboración propia.

10.2. Discusión comparativa de resultados

Las tres arquitecturas evaluadas muestran comportamientos diferenciados frente al problema de detección de crisis en ventanas de EEG, tanto en términos de métricas como de requerimientos de cómputo. El Cuadro 4, el Cuadro 6 y el Cuadro 8 permiten comparar de forma directa exactitud, sensibilidad, especificidad y métricas basadas en umbral. Las curvas ROC y Precisión–*Recall* correspondientes (Figuras 10, 11, 12, 13, 14 y 15) complementan este análisis al mostrar cómo varía el desempeño al mover el umbral de decisión.

En términos globales, la red híbrida obtiene el mejor AUC ROC (0.735), lo que indica una capacidad superior para ordenar ventanas con y sin crisis a lo largo de un rango amplio de umbrales. La TCN, por su parte, logra el mejor AP (0.264 frente a 0.225 de la híbrida y 0.103 del *Transformer*), lo cual sugiere un mejor equilibrio entre precisión y recuperación cuando se enfatiza la clase positiva en un conjunto desbalanceado. La exactitud es similar en TCN y red híbrida (0.791 y 0.770), pero la sensibilidad es mayor en la arquitectura híbrida (0.616 frente a 0.581), mientras que la TCN ofrece mayor especificidad (0.806 frente a 0.781). Esto se refleja en las matrices de confusión de los cuadros5 y 7: la TCN tiende a ser más conservadora con la clase *Crisis*, y la híbrida recupera más ventanas ictales a costa de un incremento moderado en falsos positivos.

El *Transformer* queda por detrás de ambas arquitecturas en las métricas más sensibles al desbalance. Aunque su AUC ROC (0.678) indica una capacidad discriminativa aceptable, la combinación de sensibilidad (0.344) y AP (0.103) muestra que el modelo tiene dificultades para mantener una recuperación alta sin degradar en exceso la precisión. La matriz de confusión de el Cuadro 9 evidencia que el *Transformer* acierta la mayoría de las ventanas sin crisis, pero omite una fracción considerable de ventanas ictales. En un contexto clínico, este perfil lo vuelve menos adecuado como modelo principal en comparación con la TCN y la red híbrida, salvo que se ajuste el umbral de forma agresiva o se combine con etapas adicionales de posprocesamiento.

El entorno de cómputo influyó en la forma de explorar y ajustar cada arquitectura. Los mejores modelos de TCN y red híbrida se obtuvieron en la computadora de laboratorio, donde se concentró la mayor parte de los experimentos y ajustes finos de hiperparámetros. A pesar de contar con una GPU más modesta, este entorno permitió entrenar configuraciones suficientemente ricas y repetir corridas para afinar el umbral y las estrategias de balanceo de clases. En contraste, la configuración más competitiva del *Transformer* se alcanzó en la instancia Amazon EC2, que ofreció recursos de GPU y memoria más holgados y facilitó probar variantes con mayor demanda computacional. Esto sugiere que, para arquitecturas más pesadas y sensibles a la capacidad de cómputo, como los *Transformers*, el acceso a hardware escalable es relevante para explorar su potencial, aunque en este caso no fue suficiente para superar a las arquitecturas más especializadas.

En conjunto, los resultados indican que, bajo las condiciones y datos empleados, la red híbrida ofrece el mejor compromiso cuando se prioriza la recuperación de eventos ictales, mientras que la TCN resulta ventajosa cuando se busca reducir falsos positivos y mantener un AP elevado. El *Transformer* aporta una referencia valiosa sobre el comportamiento de modelos de atención pura en este dominio, pero requeriría ajustes adicionales de arquitectura, regularización o mayor cantidad de datos para resultar competitivo frente a las otras

dos propuestas.

10.3. Resultados de optimización del flujo de entrenamiento

En esta sección se presentan los resultados del análisis comparativo de los flujos de entrenamiento implementados en PyTorch y TensorFlow, contrastados con la *pipeline* original en MATLAB desarrollada en la fase previa. Para aislar el efecto de las optimizaciones, se utilizó en los tres entornos una red TCN análoga, con la misma configuración de datos y de hiperparámetros. Primero se evaluó el impacto de las mejoras introducidas en cada *pipeline* sobre el tiempo de entrenamiento por época. Luego, estas variantes se ejecutaron en los distintos entornos de hardware descritos en el Cuadro 3, con el fin de comparar cómo se combinan las ganancias debidas al software con las debidas a la infraestructura de cómputo.

10.3.1. Comparación de entrenamiento con la fase pasada

Cuadro 10. Comparación de tiempos por época entre distintas plataformas y bibliotecas

Plataforma	**Horas/época**		
	TensorFlow	**PyTorch**	**MATLAB**
HPC	6	7	19
Amazon EC2	4	5	–

Nota. La tabla muestra el tiempo aproximado requerido por época para entrenar la red TCN utilizando distintas bibliotecas de Python y MATLAB sobre dos plataformas de cómputo (HPC y Amazon EC2). Elaboración propia.

El Cuadro 10 evidencia una reducción sustancial de tiempo por época al migrar el entrenamiento de MATLAB a las *pipelines* en Python. En la HPC, el entrenamiento con TCN pasó de aproximadamente 19 horas por época en MATLAB a 6 horas con TensorFlow y 7 con PyTorch, lo que representa una mejora de un orden de magnitud en tiempo de cómputo para una arquitectura análoga. En la instancia Amazon EC2, donde no se corrieron experimentos en MATLAB, TensorFlow y PyTorch reducen todavía más el tiempo por época (4 y 5 horas, respectivamente), aprovechando mejor la combinación de GPU y entorno optimizado para cómputo en la nube.

Las diferencias entre TensorFlow y PyTorch son moderadas, con TensorFlow ligeramente más rápido en ambos entornos. Este comportamiento es consistente con el uso de precisión mixta y compilación a grafo en TensorFlow, que permite fusionar operaciones y reducir sobrecarga, mientras que en PyTorch el flujo se beneficia más de optimizaciones como `torch.compile` pero mantiene un coste adicional asociado a la ejecución más flexible. En conjunto, los resultados muestran que el impacto de las optimizaciones de *pipeline* es comparable al efecto de cambiar de plataforma: tanto el paso de MATLAB a Python como el salto de HPC a EC2 se traducen en reducciones claras del tiempo total de entrenamiento.

Para complementar el análisis de tiempos, el Cuadro 11 resume el desempeño de la

TCN en términos de precisión y proporción de verdaderas crisis y verdaderas no crisis para cada combinación de plataforma y biblioteca. Esto permite verificar que las ganancias de velocidad no se obtuvieron a costa de degradar la capacidad de detección.

Cuadro 11. Resumen de desempeño de la TCN por plataforma y biblioteca

Plataforma	Biblioteca	Exactitud (*accuracy*)	% verdaderas crisis (TP)	% verdaderas no crisis (TN)
HPC	TensorFlow	85	30	82
HPC	PyTorch	83	20	81
HPC	MATLAB	81	18	81
Amazon EC2	TensorFlow	83	27	80
Amazon EC2	PyTorch	82	17	83

Nota. La tabla resume, para la TCN análoga utilizada en todas las evaluaciones, la exactitud global (*accuracy*) y los porcentajes de verdaderas crisis (ventanas ictales correctamente detectadas, TP) y verdaderas no crisis (ventanas sin crisis correctamente rechazadas, TN) en cada combinación de plataforma y biblioteca. Los valores correspondientes a MATLAB se tomaron de la fase previa del proyecto, mientras que los de TensorFlow y PyTorch provienen de los entrenamientos realizados en esta fase. Elaboración propia.

Al analizar el Cuadro 11 se observa que, para una misma plataforma, las diferencias entre TensorFlow y PyTorch se mantienen dentro de un rango acotado. En la HPC, TensorFlow alcanza la mayor exactitud (85 %), con un 30 % de verdaderas crisis y 82 % de verdaderas no crisis, mientras que PyTorch obtiene 83 % de exactitud con 20 % de verdaderas crisis y 81 % de verdaderas no crisis. MATLAB se sitúa ligeramente por debajo, con 81 % de exactitud y porcentajes de verdaderas crisis y verdaderas no crisis de 18 % y 81 %, respectivamente, lo que confirma que la migración a Python mantiene o mejora la capacidad de detección de la TCN.

En Amazon EC2, los resultados de TensorFlow y PyTorch son comparables a los de la HPC: TensorFlow logra 83 % de exactitud con 27 % de verdaderas crisis y 80 % de verdaderas no crisis, mientras que PyTorch obtiene 82 % de exactitud con 17 % de verdaderas crisis y 83 % de verdaderas no crisis. Estas cifras indican que las mejoras en flujo de entrenamiento y en hardware reducen de forma notable el tiempo por época (Cuadro 10) sin sacrificar el desempeño en términos de detección de crisis ni de rechazo de segmentos sin evento, cumpliendo el objetivo de acelerar el entrenamiento manteniendo la calidad de las predicciones.

10.3.2. Comparación de entre redes y equipo utiilzado

Los resultados de velocidad por muestra analizada de el Cuadro 12 muestran dos tendencias claras. En primer lugar, para una misma plataforma, la TCN es sistemáticamente la arquitectura más rápida, seguida por la red híbrida y, por último, el *Transformer*. Esta diferencia es coherente con la complejidad relativa de cada modelo: la red híbrida incorpora bloques adicionales y el *Transformer* basa todo su procesamiento en capas de autoatención, lo que incrementa el costo computacional por muestra. En segundo lugar, entre plataformas, la instancia Amazon EC2 presenta los menores tiempos promedio por muestra, en particular para la TCN (168 ms/muestra), mientras que la PC de laboratorio ocupa una posición intermedia y la HPC resulta la más lenta según esta métrica, a pesar de disponer de más memoria RAM y VRAM.

Cuadro 12. Comparación de velocidad por muestra analizada (*ms/muestra*) entre plataformas y arquitecturas

Plataforma	Especificación	TCN (ms/muestra)	Híbrida (ms/muestra)	Transformer (ms/muestra)
PC Tradicional	32 GB RAM, NVIDIA T400 (4 GB VRAM)	325	467	635
HPC	256 GB RAM, NVIDIA Quadro P5000 (16 GB VRAM)	828	1045	1265
Amazon EC2	64 GB RAM, NVIDIA Tesla T4 (16 GB VRAM)	168	213	432

Nota. La tabla resume el tiempo promedio requerido por muestra analizada (*ms/muestra*) para cada arquitectura (TCN, híbrida y Transformer) en distintas plataformas de cómputo, considerando las características de hardware disponibles. El valor de *ms/muestra* se obtuvo dividiendo el tiempo total de entrenamiento entre el número de muestras procesadas, por lo que valores menores indican una mayor velocidad de procesamiento. Elaboración propia.

Esta inversión respecto a lo esperable por potencia teórica se explica en buena medida por las diferencias en la *compute capability* de las GPU empleadas (Cuadro 13). La PC de laboratorio y la instancia Amazon EC2 utilizan GPUs con *compute capability* 7.5, compatibles con precisión mixta basada en *Tensor Cores* y con estrategias modernas de compilación como `torch.compile` y `tf.function`/XLA. En cambio, la GPU de la HPC (Quadro P5000) tiene *compute capability* 6.1, por debajo del umbral requerido para aprovechar plenamente estos mecanismos. Como consecuencia, en la HPC el entrenamiento debió realizarse en precisión simple (FP32) y con ejecución más cercana al modo imperativo, sin los beneficios de fusión agresiva de operaciones ni del aceleramiento específico para operaciones mixtas, lo que limita el rendimiento efectivo a pesar de la mayor memoria disponible.

Cuadro 13. GPU y *compute capability* por plataforma utilizada

Plataforma	**GPU**	***Compute capability***
PC Tradicional	NVIDIA T400	7.5
HPC	NVIDIA Quadro P5000	6.1
Amazon EC2	NVIDIA Tesla T4	7.5

Nota. La tabla muestra la GPU principal empleada en cada plataforma y su *compute capability* declarada. Las GPUs con *compute capability* ≥ 7.0 permiten aprovechar de forma más completa la precisión mixta basada en *Tensor Cores* y las estrategias de compilación a grafo utilizadas en las *pipelines* de *PyTorch* y *TensorFlow*, mientras que la GPU de la HPC (6.1) quedó limitada a ejecución en precisión simple (FP32) y sin estas optimizaciones avanzadas. Elaboración propia.

El Cuadro 14 complementa este análisis al mostrar el tiempo total necesario para entrenar los modelos que alcanzaron los mejores resultados en cada arquitectura. La TCN requirió 26 épocas y aproximadamente 12 horas en la PC de laboratorio, mientras que la red híbrida necesitó 27 épocas y 15 horas en la misma plataforma, coherente con el mayor costo por muestra reportado en el Cuadro 12. En contraste, el *Transformer* alcanzó su mejor configuración tras 23 épocas en Amazon EC2, pero con un tiempo total de 24 horas, lo que evidencia que, incluso sobre una GPU moderna con *compute capability* 7.5, el entrenamiento de arquitecturas basadas en atención sigue siendo considerablemente más pesado. En conjunto, estos resultados indican que la elección de arquitectura, la *compute capability* de la

Cuadro 14. Tiempo total de entrenamiento para los mejores resultados obtenidos

Modelo	**Épocas**	**Horas**	**Plataforma**
TCN	26	12	PC de laboratorio
Híbrida	27	15	PC de laboratorio
Transformer	23	24	Amazon EC2

Nota. La tabla muestra la duración total de los entrenamientos que alcanzaron los mejores resultados en cada arquitectura, indicando el número de épocas, el tiempo requerido y la plataforma utilizada. Elaboración propia.

GPU y las optimizaciones disponibles en cada entorno afectan tanto las métricas de detección como la viabilidad práctica del entrenamiento: TCN e híbrida ofrecen un compromiso más favorable entre tiempo de cómputo y desempeño, mientras que el *Transformer* exige recursos y tiempos de entrenamiento más elevados para resultados que, en este estudio, no superan a las alternativas más ligeras.

CAPÍTULO 11

Actualización de la herramienta

En este capítulo se documenta la actualización de la herramienta desarrollada en la línea de investigación, con el objetivo de integrar de forma práctica los modelos de aprendizaje profundo entrenados y el nuevo *pipeline* de análisis de EEG. Primero se presenta la API web que expone como servicio el flujo completo de preprocesamiento, extracción de características y predicción, separando la lógica de inferencia del entorno de desarrollo. Luego se describe el módulo de predicción incorporado en la herramienta de HUMANA en MATLAB, que permite enviar estudios EDF al servicio y visualizar las detecciones sobre el trazado EEG, así como una interfaz de usuario en Python que actúa como cliente ligero del API. De esta manera, se establece un puente entre los experimentos realizados en esta fase y su uso operativo en entornos clínicos o de investigación, garantizando consistencia y trazabilidad en el análisis de registros reales.

11.1. Interfaz de programación de aplicaciones para el análisis de EEG

Una interfaz de programación de aplicaciones (*Application Programming Interface*, API) es un conjunto de reglas y contratos que permite a distintos programas comunicarse entre sí sin exponer los detalles internos de implementación. En el contexto de servicios web, una API suele ofrecer un conjunto de operaciones accesibles mediante peticiones HTTP que reciben datos estructurados, ejecutan una lógica de procesamiento y devuelven respuestas en un formato estándar, típicamente JSON. De este modo, aplicaciones escritas en lenguajes distintos pueden reutilizar la misma funcionalidad de análisis, siempre que respeten el contrato de entrada y salida definido por la API [68].

En este trabajo se desarrolló una API web que actúa como fachada del flujo completo de

preprocesamiento, extracción de características y evaluación de las redes entrenadas para la detección de convulsiones. El servicio permite enviar un registro de EEG en formato EDF, acompañado opcionalmente por un archivo CSV con etiquetas de eventos, y devuelve un conjunto de resultados que incluyen un resumen de los canales leídos, la lista de eventos de convulsión detectados y, cuando se proporcionan etiquetas, métricas de desempeño del modelo. El servidor carga internamente los modelos ya entrenados (en versiones equivalentes para TensorFlow y PyTorch) y aplica de forma automática el mismo *pipeline* utilizado en los experimentos: proyección al montaje bipolar seleccionado, filtrado pasa banda y de muesca, remuestreo, normalización, ventaneo deslizante y cálculo de características adicionales cuando corresponde.

La API expone operaciones específicas para dos usos principales. En primer lugar, el análisis exploratorio de un registro individual, en el que a partir de un único EDF se obtienen las amplitudes normalizadas, la estructura de canales y la detección automática de segmentos ictales. En segundo lugar, la evaluación de los modelos entrenados sobre registros con anotaciones, lo que reproduce el flujo de evaluación utilizado en el desarrollo: el servicio aplica el modelo sobre las ventanas etiquetadas, calcula probabilidades de crisis, escanea un rango de umbrales de decisión y selecciona aquel que optimiza una métrica objetivo, como la puntuación F_1. A partir de este umbral se construye la matriz de confusión y se derivan métricas agregadas como exactitud, sensibilidad, precisión y áreas bajo las curvas ROC y Precisión–*Recall*, que se devuelven en la respuesta junto con los eventos detectados.

Este diseño desacopla el uso de los modelos del entorno de investigación en el que fueron entrenados. Cualquier cliente capaz de emitir peticiones HTTP y enviar archivos EDF puede aprovechar la API sin conocer los detalles de TensorFlow, PyTorch ni del preprocesamiento interno. Al mismo tiempo, la API garantiza que el análisis siga exactamente las mismas configuraciones de montaje, filtros, ventanas y umbrales documentadas en esta fase del proyecto, lo que facilita la reproducibilidad de resultados y la integración futura con herramientas clínicas o de investigación que requieran automatizar la detección de convulsiones.

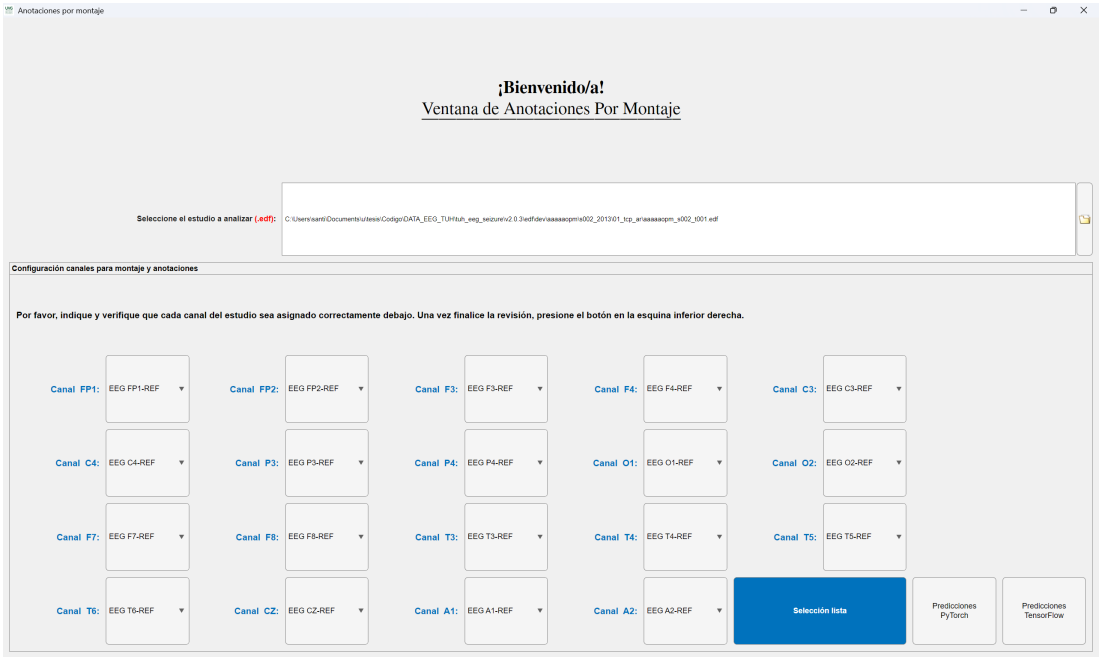
11.2. Módulo de predicción integrado en la herramienta

La herramienta de HUMANA ha evolucionado en varias fases. En la etapa previa, Pérez [18] consolidó la migración hacia un ejecutable independiente y actualizó la interfaz para facilitar el trabajo con estudios EEG bajo el sistema 10–20, integrando montajes bipolares y vistas multicanal orientadas al uso clínico. Sobre esa base se desarrolló en esta fase un módulo específico de predicción que conecta la interfaz gráfica con los modelos de aprendizaje profundo entrenados.

El módulo se organiza alrededor de una ventana en la que el usuario carga un estudio en formato EDF, selecciona el conjunto de canales que se proyectarán al montaje bipolar utilizado por las redes y configura los parámetros básicos de visualización. Una vez preparado el estudio, la interfaz ofrece dos botones de predicción que permiten enviar el registro al servicio web, eligiendo si se utilizará la versión del modelo desplegada en PyTorch o la versión equivalente en TensorFlow. La Figura 16 ilustra esta interfaz, donde se observan los controles de selección de canales, el montaje y los botones que disparan la consulta al API.

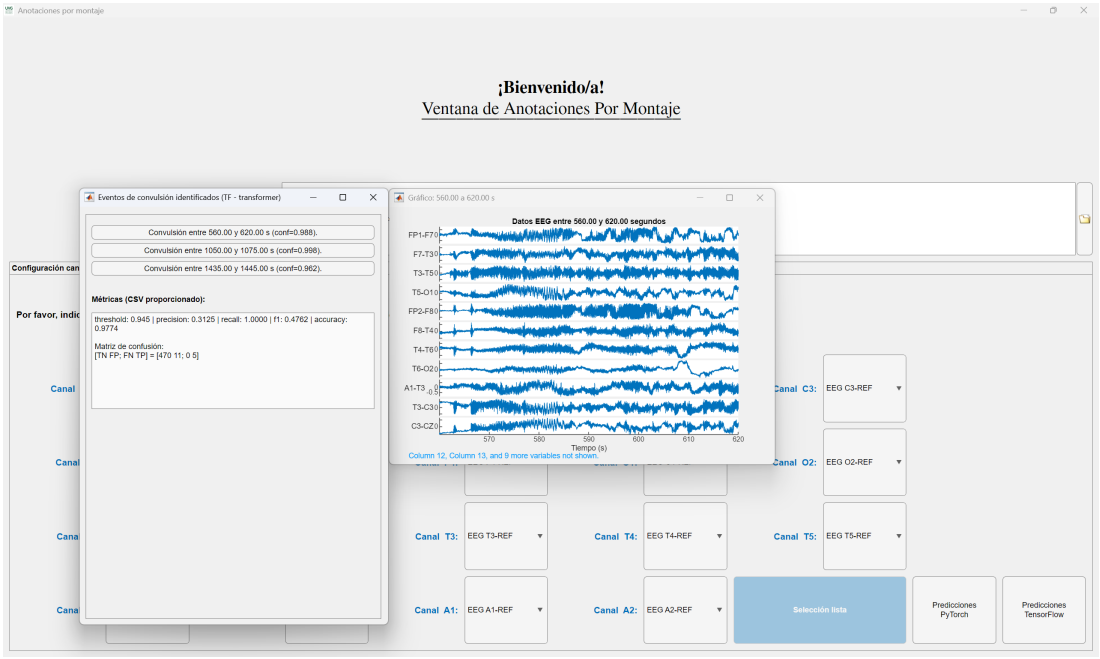
Al activar cualquiera de los botones de predicción, la herramienta empaqueta el EDF con la configuración de montaje y preprocesamiento requerida y envía la solicitud al API. El servicio aplica el mismo *pipeline* documentado en este trabajo, produce las probabilidades de crisis por ventana y devuelve los segmentos de interés junto con un resumen de métricas cuando se dispone de anotaciones de referencia. Si únicamente se cuenta con el EDF, el resultado se muestra como ventanas resaltadas sobre el trazado EEG. Cuando en el mismo directorio se encuentra además un archivo CSV con anotaciones, la respuesta incluye medidas como sensibilidad, especificidad y precisión por ventana, que permiten estimar la confiabilidad de la predicción frente a las etiquetas clínicas. La Figura 17 muestra un ejemplo de esta visualización, donde se combinan los eventos convulsivos detectados, las métricas calculadas por el API y la representación gráfica del segmento EEG asociado al intervalo consultado.

Figura 16. Interfaz gráfica para selección de canales y solicitud de predicciones



Nota: La imagen muestra la interfaz desarrollada para configurar el montaje, seleccionar los canales de un registro EEG y enviar la solicitud de predicciones a los modelos implementados (PyTorch o TensorFlow). Elaboración propia.

Figura 17. Visualización de resultados obtenidos desde la consulta al API



Nota: Se ilustran los eventos convulsivos detectados por el modelo, las métricas entregadas por el API y la representación gráfica del segmento EEG asociado al intervalo consultado. Elaboración propia.

11.3. Interfaz de usuario en Python para interacción con el API

Además de la integración con la herramienta desarrollada en MATLAB, se implementó una interfaz de usuario independiente en Python que actúa como cliente ligero del API de predicción. Esta interfaz permite ejecutar el flujo completo de análisis sin necesidad de abrir el entorno de desarrollo ni manejar directamente las rutas de los modelos o de la base de datos, de modo que el usuario interactúa únicamente mediante controles sencillos.

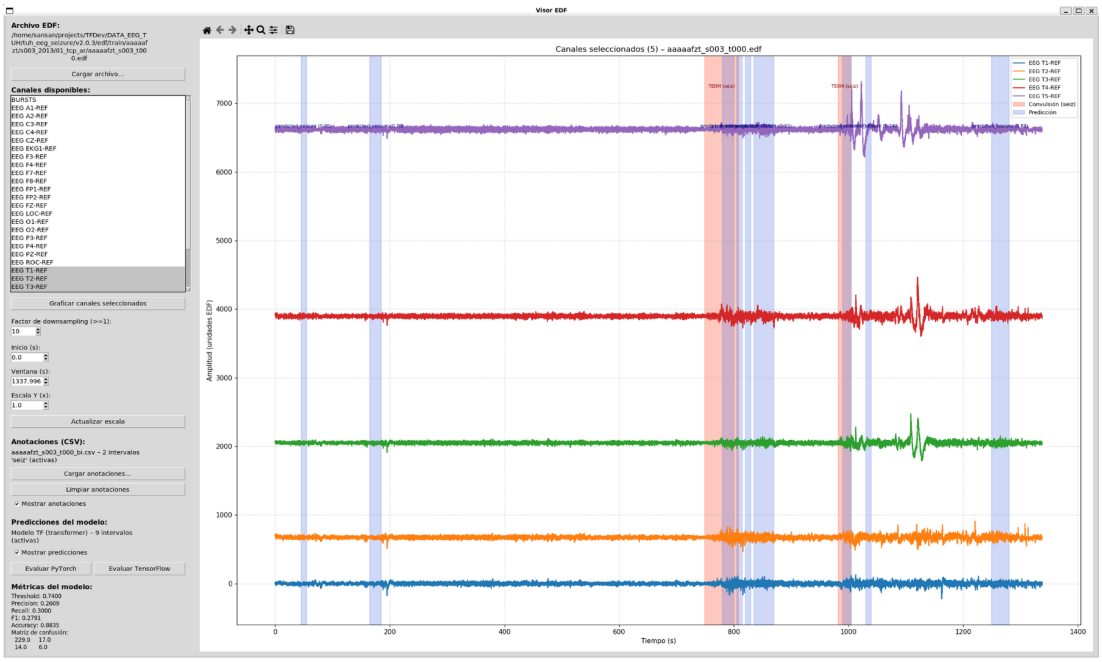
La interfaz se organiza en torno a una vista principal en la que el usuario selecciona el archivo EDF que desea analizar y, de forma opcional, un archivo CSV con anotaciones de referencia para evaluación. En el mismo panel se elige el modelo de inferencia que se va a utilizar, indicando si la solicitud se enviará a la versión desplegada en PyTorch o a la versión equivalente en TensorFlow. La configuración de parámetros básicos, como el montaje por defecto y el umbral inicial de decisión, se realiza mediante campos de entrada y controles desplegables preconfigurados, de manera que el usuario no necesita editar archivos de configuración externos. La Figura 18 ilustra el funcionamiento de esta interfaz, mostrando cómo se visualizan simultáneamente los canales seleccionados, las anotaciones de referencia y las predicciones del modelo sobre el mismo registro EEG.

Una vez seleccionados los archivos y el modelo, la interfaz ofrece un botón de ejecución que envía la solicitud al API. El cliente en Python se encarga de empaquetar los archivos,

construir la petición HTTP y gestionar la respuesta. Cuando el API devuelve las probabilidades de crisis y los segmentos de interés, la interfaz presenta un resumen estructurado de los resultados, incluyendo la lista de intervalos detectados con sus marcas de inicio y fin, y un indicador de la probabilidad asociada a cada segmento. Si el usuario proporcionó anotaciones en formato CSV, la respuesta incluye además las métricas calculadas por el servicio (por ejemplo, sensibilidad, especificidad, precisión y puntuación F_1), que se muestran en un panel de resultados junto a un resumen textual que facilita la interpretación de la confiabilidad de la predicción.

La interfaz incorpora también mecanismos básicos de registro y trazabilidad. Cada ejecución conserva en memoria la configuración utilizada, el identificador del modelo al que se consultó y un resumen de las métricas obtenidas, lo que permite repetir análisis con parámetros similares o comparar de forma informal diferentes modelos sobre el mismo registro. Al estar implementada en Python, la interfaz puede ejecutarse en entornos de escritorio o integrarse en aplicaciones web ligeras, reutilizando las mismas dependencias y versiones de bibliotecas que el resto del *pipeline* de entrenamiento y evaluación, y garantizando así un comportamiento consistente entre la fase de investigación y el uso práctico de los modelos.

Figura 18. Visor de EEG en Pythoncon anotaciones y predicciones del modelo



Nota: La figura muestra la herramienta desarrollada para visualizar registros EDF en Python, seleccionar canales, cargar anotaciones y superponer las predicciones del modelo sobre la señal EEG. Se ilustran intervalos anotados (rojo) y predicciones automáticas (azul) sobre varios canales en paralelo. Elaboración propia.

CAPÍTULO 12

Conclusiones

- Al aplicar los algoritmos de aprendizaje profundo a un volumen sustancial de ventanas de EEG y evaluar su desempeño con métricas centradas en la sensibilidad para la detección de ventanas ictales, se observó un patrón consistente: a mayor cantidad y diversidad de datos, los modelos aprendieron más por época y alcanzaron una mayor sensibilidad en la identificación de segmentos de convulsión. Este efecto se hizo evidente al comparar corridas con diferentes tamaños de conjunto y al mantener constantes el resto de hiperparámetros del entrenamiento
- Se migró el flujo de trabajo desde MATLAB hacia Python (TensorFlow y PyTorch), lo cual permitió reducir de forma importante los tiempos de entrenamiento por época mediante pipelines optimizados, precisión mixta y mejor aprovechamiento de GPU. Esto dejó una base más flexible y reproducible para continuar la línea de investigación.
- Se implementaron y compararon tres arquitecturas profundas (TCN, híbrida CNN + RNN con atención y Transformer temporal) para la detección de segmentos ictales en EEG multi-paciente. La TCN y la red híbrida mostraron el mejor compromiso entre sensibilidad, especificidad y costo computacional, mientras que el Transformer resultó más costoso de entrenar y con menor desempeño en este escenario desbalanceado.
- El análisis estadístico de las métricas (exactitud, sensibilidad, especificidad, precisión, F1, AUC ROC y AP), junto con las curvas ROC y Precisión–*Recall*, mostró que los modelos son capaces de priorizar segmentos candidatos a crisis, pero la precisión por ventana sigue siendo limitada. Esto refuerza la necesidad de seguir ajustando umbrales y estrategias de posprocesamiento orientadas a episodios completos y no solo a ventanas aisladas.
- Se observó que aumentar el volumen y la diversidad de datos —en número de ventanas y de pacientes— mejora la capacidad de los modelos para identificar patrones ictales. Al mismo tiempo, se comprobó que el fuerte desbalance de clases sigue siendo un factor crítico y que es necesario controlar cuidadosamente la procedencia de los datos para evitar sesgos de dominio y preservar la generalización entre pacientes.

- La comparación entre recursos de cómputo (PC de laboratorio, HPC y Amazon EC2) evidenció que los recursos y la *compute capability* de la GPU influyen de forma directa en la viabilidad práctica de los entrenamientos y en la velocidad por muestra. En el contexto del trabajo, las arquitecturas TCN e híbrida ofrecieron mejores tiempos y resultados que el Transformer, lo que las hace más adecuadas para iterar sobre conjuntos de datos crecientes.

- Se actualizó la herramienta de HUMANA incorporando una API web que expone el pipeline de análisis como servicio y clientes en MATLAB y Python para solicitar predicciones sobre registros EDF. Esta integración permite reutilizar los modelos sin depender del entorno de desarrollo, facilita la visualización conjunta de señales, anotaciones y salidas de los modelos, y deja una infraestructura lista para extenderse a nuevos estudios o escenarios de validación clínica.

- En conjunto, el trabajo consolidó una plataforma técnica moderna para el análisis de EEG con aprendizaje profundo, obtuvo modelos funcionales capaces de señalar segmentos de interés en crisis epilépticas y caracterizó los límites actuales impuestos por los datos disponibles y el desbalance. Los resultados deben entenderse como una base sólida sobre la cual incorporar más datos anotados, técnicas avanzadas de balanceo y posprocesamiento, y estudios futuros en contextos clínicos controlados.

- El uso de características por ventana mejoró la separabilidad entre clases cuando la cantidad de datos era limitada, permitiendo obtener modelos útiles con ejemplos anotados. Sin embargo, al incrementar el volumen y la diversidad de datos, parte de ese beneficio se diluyó e incluso algunas configuraciones introdujeron sesgos hacia ciertos patrones frecuentes, lo que sugiere que estas características son especialmente valiosas en escenarios de pocos datos o de ajuste fino, pero deben revisarse críticamente cuando se escala a cantidades de datos mayores.

- El trabajo evidenció la importancia de cuidar los aspectos metodológicos para no sobreestimar el desempeño: se debe evitar la fuga de información entre particiones, aplicar normalización de forma coherente usando exclusivamente estadísticas del conjunto de entrenamiento y evaluar siempre en un esquema entre pacientes. Bajo estas restricciones más realistas, las métricas reflejan mejor el comportamiento esperado de los modelos en un uso clínico real.

CAPÍTULO 13

Recomendaciones

- Se sugiere aumentar la proporción de ejemplos positivos de convulsión incorporando varios conjuntos de datos públicos con etiquetas clínicas (por ejemplo, CHB-MIT). Una muestra más amplia y diversa de pacientes, montajes y condiciones puede mejorar la generalización del modelo y reducir la dependencia de técnicas de re-muestreo como el submuestreo de la clase mayoritaria o el sobremuestreo de la minoritaria. Para lograrlo, será necesario armonizar formatos y criterios de etiquetado entre bases y reportar resultados por paciente y, cuando sea posible, entre conjuntos, con el fin de controlar diferencias entre fuentes y demostrar robustez.

- Explorar codificadores y decodificadores aprendidos (como autoencodificadores o bloques de proyección) para transformar el espacio de características antes de la clasificación. Un codificador puede reducir la dimensionalidad y filtrar ruido, dejando una representación más separable entre clases; un decodificador ayuda a conservar la información relevante durante el entrenamiento. En EEG, un bloque de codificación previo al clasificador puede destilar patrones entre canales y en el tiempo, lo que mitiga el desbalance al clarificar las fronteras de decisión sin depender en exceso del re-muestreo. Para resultados consistentes, combinar esta estrategia con ponderación por clase o funciones de pérdida adaptadas.

- Explorar aprendizaje por refuerzo para realizar ajuste fino del modelo hacia los casos de convulsión. Además de la pérdida habitual, el modelo recibiría una recompensa que valore más detectar episodios ictales y que penalice con mayor peso los falsos negativos. Este esquema puede ayudar a enfocar el aprendizaje en los eventos relevantes sin depender tanto del re-muestreo, siempre que la señal de recompensa se diseñe con criterios clínicos y que la evaluación se haga por paciente para verificar la generalización.

- Evaluar la migración de más módulos del Epileptic EEG Analysis Toolbox a Python o a otro lenguaje con compatibilidad con MATLAB para reutilizar el código existente y, al mismo tiempo, eliminar la dependencia del compilador y de la licencia de pago de MATLAB. El objetivo es disponer de una herramienta libre y de más fácil acceso para equipos u organizaciones fuera de la universidad y crear un ambiente más robusto

y seguro para la interfaz y los datos que se manejan. Esta transición debe incluir la verificación funcional frente a los resultados actuales, la armonización de formatos y la documentación de los cambios para asegurar continuidad y reproducibilidad.

- Profundizar en el análisis de características y en la selección de canales para identificar qué combinaciones de derivaciones y *features* aportan información realmente discriminativa para la detección de crisis. Esto puede incluir ingeniería de características (análisis de relevancia, ablation studies, selección secuencial, etc.) y evaluación sistemática de subconjuntos de canales. Un conjunto de características y canales más depurado permitiría entrenar modelos menos complejos y más rápidos, con menos riesgo de sobreajuste y con patrones más estables entre pacientes.

- Explorar *pipelines* híbridos donde se utilicen técnicas de aprendizaje automático o profundo como etapa de preprocesamiento para extraer representaciones de mayor nivel (por ejemplo, codificadores automáticos, modelos auto-supervisados o redes entrenadas para agrupar ventanas en patrones prototípicos), y alimentar esas representaciones a modelos posteriores más ligeros (redes neuronales poco profundas u otros clasificadores). Comparar estos esquemas en términos de desempeño, tiempo de entrenamiento, uso de memoria y estabilidad de la convergencia permitiría determinar si separar explícitamente la extracción de características de la etapa de clasificación mejora la eficiencia y la robustez.

CAPÍTULO 14

Referencias

[1] R. S. Fisher, C. Acevedo, A. Arzimanoglou, A. Bogacz, J. H. Cross, C. E. Elger, J. Engel, L. Forsgren, J. A. French, M. Glynn, D. C. Hesdorffer, B. I. Lee, G. W. Mathern, S. L. Moshé, E. Perucca, I. E. Scheffer, T. Tomson, M. Watanabe y S. Wiebe, «Definición clínica práctica de la epilepsia,» *Epilepsia*, vol. 55, págs. 475-482, 4 2014, ISSN: 15281167. DOI: 10.1111/epi.12550.

[2] J. Carlos, L. Girón, A. Antonio, J. Magaña, O. Gerardo, R. Samayoa, J. Manuel y P. Córdova, *Epilepsia Enfoque Multidisciplinario*, 2.ª ed. 2014, ISBN: 9789929404618. dirección: www.humanagt.org.

[3] World Health Assembly, «Global burden of epilepsy and the need for coordinated action at the country level to address its health, social and public knowledge implications: report by the Secretariat,» n.º 68, 2015. dirección: https://iris.who.int/handle/10665/252840.

[4] A. Ghaiyoumi, *Epilepsy*, feb. de 2024. dirección: https://www.who.int/news-room/fact-sheets/detail/epilepsy.

[5] J. E. B. del Busto, «La epilepsia, un problema de salud a escala mundial Epilepsy, a global health problem,» *Revista Habanera de Ciencias Médicas*, págs. 660-663, oct. de 2014, ISSN: 1729-519X.

[6] A. Craik, Y. He y J. L. Contreras-Vidal, *Deep learning for electroencephalogram (EEG) classification tasks: A review*, 2019. DOI: 10.1088/1741-2552/ab0ab5.

[7] N. Ke, T. Lin, Z. Lin, X.-H. Zhou y T. Ji, «Convolutional Transformer Networks for Epileptic Seizure Detection,» en *Proceedings of the 31st ACM International Conference on Information & Knowledge Management*, ép. CIKM '22, Atlanta, GA, USA: Association for Computing Machinery, 2022, págs. 4109-4113, ISBN: 9781450392365. DOI: 10.1145/3511808.3557568. dirección: https://doi.org/10.1145/3511808.3557568.

[8] M. Gunavathi, S. Sudha, S. Oviyaajanani y V. Girija, *Detecting the Seizure Conditions of Humans with EEG Dataset using Deep Belief Network Algorithm*. dirección: https://doi.org/10.56155/978-81-955020-9-7-23.

[9] M. J. Angulo, «Análisis y Reconocimiento de Patrones de Señales Biomédicas de Pacientes con Epilepsia,» Trabajo de graduación de licenciatura, Universidad del Valle de Guatemala, 2020.

[10] M. F. Pineda, «Diseño e Implementación de una Base de Datos de Señales Biomédicas de Pacientes con Epilepsia,» Trabajo de graduación de licenciatura, Universidad del Valle de Guatemala, 2020.

[11] J. D. Manrique, «Herramienta de Software con una Base de Datos Integrada para el Estudio de la Epilepsia-Fase II,» Trabajo de graduación de licenciatura, Universidad del Valle de Guatemala, 2021.

[12] D. A. Vela, «Automatización del Proceso de Anotación de Señales EEG de Pacientes con Epilepsia por Medio de Técnicas de Aprendizaje Automático,» Trabajo de graduación de licenciatura, Universidad del Valle de Guatemala, 2021.

[13] C. Lemus, «Análisis y anotación de señales bioeléctricas de pacientes con epilepsia utilizando técnicas de aprendizaje automático supervisado y no supervisado,» Trabajo de graduación de licenciatura, Universidad del Valle de Guatemala, 2022.

[14] S. J. Silvestre, «Diseño e implementación de un repositorio de acceso público de datos y señales relacionados al estudio de la epilepsia,» Trabajo de graduación de licenciatura, Universidad del Valle de Guatemala, 2022.

[15] K. V. Caceros, «Mejora y expansión de un repositorio de datos y señales biomédicas de acceso público,» Trabajo de graduación de licenciatura, Universidad del Valle de Guatemala, 2023.

[16] D. A. Méndez, «Extensión, validación y migración de una herramienta de software para el estudio de la epilepsia para su uso en el Centro de Epilepsia y Neurocirugía Funcional (HUMANA),» Trabajo de graduación de licenciatura, Universidad del Valle de Guatemala, 2023.

[17] C. I. Patzán, «Aplicación sistemática de algoritmos de aprendizaje automático para el estudio de la epilepsia y la detección de segmentos de interés en señales bioeléctricas,» Trabajo de graduación de licenciatura, Universidad del Valle de Guatemala, 2023.

[18] J. A. Pérez, «Aplicación de algoritmos de aprendizaje automático, con énfasis en aprendizaje supervisado, para la identificación y categorización de segmentos de interés en señales bioeléctricas para el estudio de la epilepsia - Fase V,» Trabajo de graduación de licenciatura, Universidad del Valle, 2024.

[19] D. A. Ixcayau, «Aplicación de algoritmos de aprendizaje automático, con énfasis en aprendizaje no supervisado, para la identicación y categorización de segmentos de interés en señales bioeléctricas para el estudio de la epilepsia - Fase V,» Trabajo de graduación de licenciatura, Universidad del Valle de Guatemala, 2024.

[20] *Epilepsia: Estudio, Tratamiento y Cirugía desde el 2006.* dirección: `https://humanagt.org/epilepsia/#:~:text=VIDEO%2DELECTROENCEFALOGRAMA`.

[21] E. C. Wirrell, R. Nabbout, I. E. Scheffer, T. Alsaadi, A. Bogacz, J. A. French, E. Hirsch, S. Jain, S. Kaneko, K. Riney, P. Samia, O. C. Snead, E. Somerville, N. Specchio, E. Trinka, S. M. Zuberi, S. Balestrini, S. Wiebe, J. H. Cross, E. Perucca, S. L. Moshé y P. Tinuper, «Methodology for classification and definition of epilepsy syndromes with list of syndromes: Report of the ILAE Task Force on Nosology and Definitions,» *Epilepsia*, vol. 63, págs. 1333-1348, 6 jun. de 2022, ISSN: 0013-9580. DOI: 10.1111/epi.17237.

[22] R. S. Fisher, J. H. Cross, J. A. French, N. Higurashi, E. Hirsch, F. E. Jansen, L. Lagae, S. L. Moshé, J. Peltola, E. R. Perez, I. E. Scheffer y S. M. Zuberi, «Operational classification of seizure types by the International League Against Epilepsy: Position Paper of the ILAE Commission for Classification and Terminology,» *Epilepsia*, vol. 58, págs. 522-530, 4 abr. de 2017, ISSN: 0013-9580. DOI: 10.1111/epi.13670.

[23] R. S. Fisher, J. H. Cross, C. D'Souza, J. A. French, S. R. Haut, N. Higurashi, E. Hirsch, F. E. Jansen, L. Lagae, S. L. Moshé, J. Peltola, E. R. Perez, I. E. Scheffer, A. Schulze-Bonhage, E. Somerville, M. Sperling, E. M. Yacubian y S. M. Zuberi, «Instruction manual for the ILAE 2017 operational classification of seizure types,» *Epilepsia*, vol. 58, págs. 531-542, 4 abr. de 2017, ISSN: 0013-9580. DOI: 10.1111/epi.13671.

[24] S. U. Fredes, A. D. Firoozabadi, P. Adasme, D. Zabala-Blanco, P. P. Játiva y C. Azurdia-Meza, «Enhanced Epileptic Seizure Detection through Wavelet-Based Analysis of EEG Signal Processing,» *Applied Sciences (Switzerland)*, vol. 14, 13 jul. de 2024, ISSN: 20763417. DOI: 10.3390/app14135783. dirección: https://www.mdpi.com/2076-3417/14/13/5783.

[25] R. Madou, M. A. Haberman, E. M. Spinelli y A. C. Ceriani, «Señales bioeléctricas del cuerpo: de la ingeniería electrónica a la performance artística,» *¡Cuerpo, Máquina, Acción!*, vol. 4, 2020.

[26] «¿Qué es un electroencefalograma?» *Clínica Universiadad de Navarra*, dirección: www.cun.es/enfermedades-tratamientos/pruebas-diagnosticas/electroencefalograma.

[27] P. Boonyakitanont, A. Lek-uthai, K. Chomtho y J. Songsiri, *A review of feature extraction and performance evaluation in epileptic seizure detection using EEG*, 2019. arXiv: 1908.00492 [eess.SP]. dirección: https://arxiv.org/abs/1908.00492.

[28] M. X. Cohen, *Analyzing neural time series data: theory and practice*. MIT press, 2014.

[29] I. Mezić, *Analysis of fluid flows via spectral properties of the koopman operator*, ene. de 2013. DOI: 10.1146/annurev-fluid-011212-140652.

[30] M. O. Williams, I. G. Kevrekidis y C. W. Rowley, «A Data-Driven Approximation of the Koopman Operator: Extending Dynamic Mode Decomposition,» ago. de 2014. DOI: 10.1007/s00332-015-9258-5. dirección: http://arxiv.org/abs/1408.4408%20http://dx.doi.org/10.1007/s00332-015-9258-5.

[31] Mauroy, I. Mezić e Y. Susuki, *Koopman Operator in Systems and Control: Concepts, Methodologies, and Applications*. Springer International Publishing, 2020.

[32] I. Goodfellow, Y. Bengio y A. Courville, *Deep Learning*. MIT Press, 2016. dirección: http://www.deeplearningbook.org.

[33] C. M. Bishop, *Pattern Recognition and Machine Learning*. Springer New York, 2006. dirección: https://www.microsoft.com/en-us/research/wp-content/uploads/2006/01/Bishop-Pattern-Recognition-and-Machine-Learning-2006.pdf.

[34] T.-Y. Lin, P. Goyal, R. Girshick, K. He y P. Dollár, «Focal loss for dense object detection,» en *Proceedings of the IEEE international conference on computer vision*, 2017, págs. 2980-2988.

[35] S. S. M. Salehi, D. Erdogmus y A. Gholipour, «Tversky loss function for image segmentation using 3D fully convolutional deep networks,» en *International workshop on machine learning in medical imaging*, Springer, 2017, págs. 379-387.

[36] D. P. Kingma y J. Ba, *Adam: A Method for Stochastic Optimization*, 2015. arXiv: 1412.6980 [cs.LG]. dirección: https://arxiv.org/abs/1412.6980.

[37] I. Loshchilov y F. Hutter, *Decoupled Weight Decay Regularization*, 2019. arXiv: 1711.05101 [cs.LG]. dirección: https://arxiv.org/abs/1711.05101.

[38] D. Hendrycks y K. Gimpel, «Gaussian error linear units (gelus),» *arXiv preprint arXiv:1606.08415*, 2016.

[39] R. T. Schirrmeister, J. T. Springenberg, L. D. J. Fiederer, M. Glasstetter, K. Eggensperger, M. Tangermann, F. Hutter, W. Burgard y T. Ball, «Deep learning with convolutional neural networks for EEG decoding and visualization,» *Human brain mapping*, vol. 38, n.º 11, págs. 5391-5420, 2017.

[40] S. Hochreiter y J. Schmidhuber, «Long Short-Term Memory,» *Neural Computation*, vol. 9, págs. 1735-1780, 8 nov. de 1997, ISSN: 08997667. DOI: 10.1162/neco.1997.9.8.1735.

[41] K. Cho, B. Van Merriënboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk e Y. Bengio, «Learning phrase representations using RNN encoder-decoder for statistical machine translation,» *arXiv preprint arXiv:1406.1078*, 2014.

[42] D. Bahdanau, K. Cho e Y. Bengio, «Neural machine translation by jointly learning to align and translate,» *arXiv preprint arXiv:1409.0473*, 2014.

[43] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser e I. Polosukhin, «Attention is all you need,» *Advances in neural information processing systems*, vol. 30, 2017.

[44] T. Saito y M. Rehmsmeier, «The precision-recall plot is more informative than the ROC plot when evaluating binary classifiers on imbalanced datasets,» *PLoS ONE*, vol. 10, 3 mar. de 2015, ISSN: 19326203. DOI: 10.1371/journal.pone.0118432.

[45] ML and DL Explained. «Understanding the Precision-Recall Curve (and Why It Matters).» Medium. Consultado el 17 de noviembre de 2025. dirección: https://medium.com/@ml_dl_explained/understanding-the-precision-recall-curve-and-why-it-matters-72297cbdbd2b.

[46] T. Fawcett, «An introduction to ROC analysis,» *Pattern Recognition Letters*, vol. 27, págs. 861-874, 8 jun. de 2006, ISSN: 01678655. DOI: 10.1016/j.patrec.2005.10.010.

[47] N. Van Otten. «ROC And AUC Curves In Machine Learning Made Simple & How To Tutorial In Python.» Spot Intelligence. Consultado el 17 de noviembre de 2025. dirección: https://spotintelligence.com/2024/06/17/roc-auc-curve-in-machine-learning/.

[48] A. Krizhevsky, I. Sutskever y G. E. Hinton, «Imagenet classification with deep convolutional neural networks,» *Advances in neural information processing systems*, vol. 25, 2012.

[49] S. Chetlur, C. Woolley, P. Vandermersch, J. Cohen, J. Tran, B. Catanzaro y E. Shelhamer, «cudnn: Efficient primitives for deep learning,» *arXiv preprint arXiv:1410.0759*, 2014.

[50] P. Micikevicius, S. Narang, J. Alben, G. Diamos, E. Elsen, D. Garcia, B. Ginsburg, M. Houston, O. Kuchaiev, G. Venkatesh et al., «Mixed precision training,» *arXiv preprint arXiv:1710.03740*, 2017.

[51] «Cuda C Programming Guide,» *NVIDIA, July*, vol. 29, n.º 31, pág. 6, 2013.

[52] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga et al., «Pytorch: An imperative style, high-performance deep learning library,» *Advances in neural information processing systems*, vol. 32, 2019.

[53] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin et al., «Tensorflow: Large-scale machine learning on heterogeneous distributed systems,» *arXiv preprint arXiv:1603.04467*, 2016.

[54] *Electroencephalography (EEG) Resources*. dirección: `https://isip.piconepress.com/projects/nedc/html/tuh_eeg/`.

[55] S. Ferrell, V. Mathew, M. Refford, V. Tchiong, T. Ahsan, I. Obeid y J. Picone, *Electrode Location and Channel Labels*, 2022.

[56] S. Rahman, S. Ferrell, I. O. Tarek Elseify y J. P. D. Ochal, *Annotation Guidelines*, jul. de 2020.

[57] *CHB-MIT Scalp EEG Database*, jun. de 2010. DOI: `https://doi.org/10.13026/C2K01R`. dirección: `https://physionet.org/content/chbmit/1.0.0/`.

[58] I. Obeid y J. Picone, «The temple university hospital EEG data corpus,» *Frontiers in Neuroscience*, vol. 10, MAY 2016, ISSN: 1662453X. DOI: `10.3389/fnins.2016.00196`.

[59] B. Kemp y J. Olivan, «European data format 'plus'(EDF+), an EDF alike standard format for the exchange of physiological data,» *Clinical neurophysiology*, vol. 114, n.º 9, págs. 1755-1761, 2003.

[60] *Reading raw data*. dirección: `https://mne.tools/stable/api/reading_raw_data.html`.

[61] W. AL-Salman, Y. Li y P. Wen, «K-complexes Detection in EEG Signals using Fractal and Frequency Features Coupled with an Ensemble Classification Model,» *Neuroscience*, vol. 422, págs. 119-133, dic. de 2019, ISSN: 18737544. DOI: `10.1016/j.neuroscience.2019.10.034`.

[62] *TFRecord and tf.train.Example*. dirección: `https://www.tensorflow.org/tutorials/load_data/tfrecord`.

[63] *Better performance with the tf.data API*. dirección: `https://www.tensorflow.org/guide/data_performance`.

[64] *XLA*. dirección: `https://openxla.org/xla`.

[65] *torch.utils.data*. dirección: `https://docs.pytorch.org/docs/stable/data.html`.

[66] *Introduction to torch.compile*. dirección: `https://docs.pytorch.org/tutorials/intermediate/torch_compile_tutorial.html`.

[67] B. Hjorth, «EEG analysis based on time domain properties,» *Electroencephalography and clinical neurophysiology*, vol. 29, n.º 3, págs. 306-310, 1970.

[68] R. T. Fielding, *Architectural styles and the design of network-based software architectures*. University of California, Irvine, 2000.

CAPÍTULO 15

Anexos

15.1. Repositorio de *pipeline* de desarrollo

- https://github.com/sansancas/TesisFinal.git

15.2. Repositorio de *pipeline* experimental

- https://github.com/sansancas/TFDev.git